ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

IMPLEMENTASI KONSEP OVERLAY NETWORK PADA GREEDY PERIMETER STATELESS ROUTING (GPSR) DI VANETs

KEVIN ARDITYA
NRP 5113 100 019

Dosen Pembimbing I
Prof. Ir. Supeno Djanali, M.Sc, Ph.D.

Dosen Pembimbing II
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

IMPLEMENTASI KONSEP OVERLAY NETWORK PADA GREEDY PERIMETER STATELESS ROUTING (GPSR) DI VANETs

KEVIN ARDITYA
NRP 5113 100 019

Dosen Pembimbing I
Prof. Ir. Supeno Djanali, M.Sc, Ph.D.

Dosen Pembimbing II
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF OVERLAY NETWORK CONCEPT ON GREEDY PERIMETER STATELESS ROUTING (GPSR) IN VANETs

KEVIN ARDITYA
NRP 5113 100 0319

Supervisor I
Prof. Ir. Supeno Djanali, M.Sc, Ph.D.

Supervisor II
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2017

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

IMPLEMENTASI KONSEP OVERLAY NETWORK PADA GREEDY PERIMETER STATELESS ROUTING (GPSR) DI VANETs

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
KEVIN ARDITYA
NRP : 5113 100 019

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Prof. Ir. Supeno Djanali, M.Sc. PHD. .....
NIP: 194806191973011001 (pembimbing 1)

Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
NIP: 198410162008121002 (pembimbing 2)

SURABAYA
JULI, 2017

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI KONSEP OVERLAY NETWORK PADA GREEDY PERIMETER STATELESS ROUTING (GPSR) DI VANETs

Nama Mahasiswa	**: KEVIN ARDITYA**
NRP	**: 5113100019**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Prof. Ir. Supeno Djanali, M.Sc, Ph.D.**
Dosen Pembimbing 2	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**

Abstrak

Vehicular Ad hoc Network (VANETs) adalah teknologi komunikasi yang menggunakan jaringan *ad hoc* untuk berkomunikasi. Pengembangan VANETs berfokus kepada *routing protocol* yang dinamis yang dapat bertahan pada lingkungan VANETs yang cepat dan tidak terprediksi. *Greedy Perimeter Stateless Routing* (GPSR) merupakan salah satu protokol dalam VANETs yang bekerja secara *geographic*. Pengiriman paket data dilakukan secara *greedy* berdasarkan posisi node tetangga menuju node *destination.* Oleh karena itu, dibutuhkan proses *route discovery* agar GPSR bisa mendeteksi ketersediaan rute menuju node *destination*.

GODV adalah modifikasi dari GPSR yang mengadopsi proses *route discovery* pada *Dynamic Source Routing* (DSR). Sebelum melakukan pengiriman paket data, node sumber mengirimkan paket *route request* menuju node *destination*. Lalu node *destination* akan merespon dengan mengirimkan paket *route reply* kembali node sumber dengan menyimpan titik-titik rute pengiriman paket data yang digunakan sebagai implementasi konsep *overlay network*.

Hasil pengujian yang dilakukan menunjukkan bahwa GODV memberikan nilai rata-rata *packet delivery ratio* (PDR) yang lebih baik dibandingkan dengan GPSR dengan kenaikan rata-rata PDR sebesar 40 %.

Kata kunci : Overlay Network, DSR, GPSR, VANETs

[Halaman ini sengaja dikosongkan]

IMPLEMENTATION OF OVERLAY NETWORK CONCEPT ON GREEDY PERIMETER STATELESS ROUTING (GPSR) IN VANETs

Student's Name	**: KEVIN ARDITYA**
Student's ID	**: 5113100019**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Prof. Ir. Supeno Djanali, M.Sc, Ph.D.**
Second Advisor	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**

Abstract

The Vehicular Ad hoc Network (VANETs) is a communication technology that use an ad hoc networks to communicate. The main focus on VANETs development is to make a dynamic routing protocol that can survive in VANETs environment which is fast and unpredictable. Greedy Perimeter Stateless Routing (GPSR) is one of VANET's protocol which is geographically based. Data packets greedily forwarded based on neighbors node position toward the destination. Hence, route discovery process is necessary to detect route availability toward the destination.

GODV is a modification of GPSR that adopt the route discovery process of Dynamic Source Routing (DSR). Before sending data packet, the source node broadcast route request to find the destination node. Then it responds by sending back a route reply to the soure node while recording all the point of routes that will be used as an implementation of overlay network concept for data packet forwarding.

The experiment result show that the average packet delivery ratio (PDR) of GODV improves 40% than GPSR.

Keyword : Overlay Network, DSR, GPSR, VANETs

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur penulis sampaikan kepada Allah SWT yang tidak henti-hentinya melimpahkan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir ini dengan judul "Implementasi Konsep *Overlay Network* pada *Greedy Perimeter Stateless Routing* (GPSR) di VANETs".

Pengerjaan Tugas Akhir ini merupakan syarat guna mendapatkan gelar Sarjana Komputer. Terselesaikannya buku Tugas Akhir ini, tidak lepas dari bantuan dan dukungan dari berbagai pihak. Penulis ingin mengucapkan terima kasih kepada:

1. Allah SWT yang telah melimpahkan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Keluarga penulis yang senantiasa memberi dukungan sehingga penulis termotivasi untuk mengerjakan Tugas Akhir ini.
3. Bapak Prof. Ir. Supeno Djanali, M.Sc, Ph.D. selaku dosen pembimbing pertama dan dosen wali penulis yang telah membantu penulis dalam menyelesaikan Tugas Akhir ini.
4. Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc., selaku dosen pembimbing kedua yang senantiasa membimbing dan memberikan arahan kepada penulis dalam mengerjakan Tugas Akhir ini.
5. Seluruh dosen dan karyawan Teknik Informatika ITS yang telah membimbing dan memberikan ilmunya kepada penulis.
6. Teman-teman seperjuangan, Bagus, Sumitra, dan Bagas yang saling mendukung dan menghibur dalam pengerjaan Tugas Akhir ini.
7. Teman-teman TC 13 yang sudah membantu penulis dalam menjalani masa perkuliahan di Teknik Informatika ITS sejak pertama kali di terima di ITS hingga akhir masa kuliah.
8. Serta semua pihak yang tidak bisa disebutkan satu per satu yang turut membantu dan memotivasi penulis dalam menyelesaikan Tugas Akhir ini.

Tidak ada di dunia ini yang sempurna begitu juga dengan Tugas Akhir ini. Penulis menyadari Tugas Akhir ini memiliki banyak kekurangan, oleh karena itu penulis mengharapkan saran dan kritik yang membangun dari pembaca.

Surabaya, Juli 2017

Kevin Arditya

DAFTAR ISI

LEMBAR PENGESAHAN v
Abstrak vii
Abstract ix
KATA PENGANTAR xi
DAFTAR ISI xiii
DAFTAR GAMBAR xvii
DAFTAR TABEL xix
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Masalah 2
1.4 Tujuan 2
1.5 Manfaat 2
1.6 Metodologi 3
1.7 Sistematika Penulisan Laporan Tugas Akhir 4
BAB II TINJAUAN PUSTAKA 5
2.1 Vehicular Ad hoc Network (VANETs) 5
2.2 Greedy Perimeter Stateless Routing (GPSR) 6
2.3 Dynamic Source Routing (DSR) 8
2.4 Overlay Network 9
2.5 Simulation of Urban Mobility (SUMO) 10
2.6 OpenStreetMap 11
2.7 JOSM 12
2.8 AWK 12
2.9 Network Simulator 2 (NS-2) 12
2.9.1 Instalasi NS-2 13
2.9.2 Patching Protokol GPSR 13
BAB III PERANCANGAN 15
3.1 Deskripsi Umum 15
3.2 Perancangan Skenario Grid 16
3.3 Perancangan Skenario Real 17
3.4 Perancangan Simulasi pada NS-2 19
3.5 Perancangan Metrik Analisis 19

3.5.1 Packet Delivery Ratio (PDR) 19
3.5.2 End to End Delay 19
3.5.3 Routing Overhead 20
3.6 Perancangan Protokol GODV 20
BAB IV IMPLEMENTASI 23
4.1 Lingkungan Implementasi Protokol 23
4.2 Implementasi Skenario Grid 23
4.3 Implementasi Skenario Real 26
4.4 Implementasi Protokol GODV 29
4.4.1 Implementasi Header Protokol DSR pada Header GPSR 29
4.4.2 Implementasi Routing Table pada GPSR 31
4.4.3 Modifikasi Class GPSRAgent 33
4.4.4 Implementasi Route Discovery pada GPSR 34
4.4.5 Implementasi Konsep Overlay Network pada RREP 35
4.4.6 Modifikasi forwarding node pada GPSR 35
4.5 Implementasi Simulasi pada NS-2 35
4.6 Implementasi Metrik Analisis 36
4.6.1 Implementasi Packet Delivery Ratio (PDR) 36
4.6.2 Implementasi End to End Delay 36
4.6.3 Implementasi Routing Overhead 37
BAB V UJI COBA DAN EVALUASI 39
5.1 Lingkungan Uji Coba 39
5.2 Skenario Uji Coba 39
5.3 Hasil Uji Coba Skenario Grid 40
5.3.1 Hasil Packet Delivery Ratio pada Skenario Grid 40
5.3.2 Hasil End to End Delay pada Skenario Grid 44
5.3.3 Hasil Routing Overhead pada Skenario Grid 46
5.4 Hasil Uji Coba Skenario Real 49
5.4.1 Hasil Packet Delivery Ratio pada Skenario Real 49
5.4.2 Hasil End to End Delay pada Skenario Real 50
5.4.3 Hasil Routing Overhead pada Skenario Real 51
BAB VI KESIMPULAN DAN SARAN 53
6.1 Kesimpulan 53
6.2 Saran 54

DAFTAR PUSTAKA.....55
LAMPIRAN.....57
BIODATA PENULIS.....81

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

Gambar 2.1 Ilustrasi komunikasi dalam VANETs [1] 6
Gambar 2.2 Metode *greedy forwarding* [2] 7
Gambar 2.3 Metode *perimeter forwarding* [2] 8
Gambar 2.4 Mekanisme *route discovery* pada DSR [3] 9
Gambar 2.5 Konsep *overlay network* 9
Gambar 3.1 Diagram rancangan simulasi. 15
Gambar 3.2 Mekanisme pembuatan skenario *grid* 17
Gambar 3.3 Mekanisme pembuatan skenari *real* 18
Gambar 3.4 Ilustrasi protokol GODV 21
Gambar 3.5 *Flowchart* protokol GODV 22
Gambar 4.1 Peta skenario *grid* 800m x 800m 24
Gambar 4.2 *randomTrips.py* sebelum modifikasi 26
Gambar 4.3 *randomTrips.py* setelah modifikasi 26
Gambar 4.4 Cuplikan skenario *grid* pada *sumo-gui* 26
Gambar 4.5 Peta Surabaya daerah Gubeng 27
Gambar 4.6 Hasil modifikasi menggunakan JOSM 27
Gambar 4.7 Cuplikan skenario *real* pada *sumo-gui* 28
Gambar 4.8 Modifikasi union hdr_all_gpsr 30
Gambar 4.9 Pendefinisian variabel pada gpsr_packet.h 30
Gambar 4.10 Implementasi *struct* hdr_godv_data 31
Gambar 4.11 Implementasi godv_rtable.h 32
Gambar 4.12 Pendefinisian variabel pada gpsr.h 33
Gambar 4.13 Inisiasi variabel pada *class* GPSRAgent 34
Gambar 5.1 Grafik PDR pada kecepatan 10 m/s 41
Gambar 5.2 Grafik PDR pada kecepatan 15 m/s 41
Gambar 5.3 Grafik PDR pada kecepatan 20 m/s 42
Gambar 5.4 Node tidak menemukan *nexthop* 43
Gambar 5.5 Node menemukan *nexthop* 43
Gambar 5.6 Grafik *end to end delay* pada kecepatan 10 m/s 44
Gambar 5.7 Grafik *end to end delay* pada kecepatan 15 m/s 45
Gambar 5.8 Grafik *end to end delay* pada kecepatan 20 m/s 45
Gambar 5.9 Grafik *routing overhead* pada kecepatan 10 m/s 47
Gambar 5.10 Grafik *routing overhead* pada kecepatan 15 m/s ... 47

Gambar 5.11 Grafik *end to end delay* pada kecepatan 20 m/s 48
Gambar 5.12 Grafik *packet delivery ratio* pada skenario *real* 49
Gambar 5.13 Grafik *end to end delay* pada skenario *real* 50
Gambar 5.14 Grafik *routing overhead* pada skenario *real*.....51

DAFTAR TABEL

Tabel 4.1 Spesifikasi lingkungan implementasi protokol 23
Tabel 5.1 Spesifikasi perangkat uji coba.....39
Tabel 5.2 Parameter skenario pengujian 39
Tabel 5.3 Simulasi pada node 50 dengan kecepatan 15 m/s 42

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dijelaskan mengenai beberapa hal dasar dalam Tugas Akhir ini yang meliputi latar belakang, perumusan masalah, batasan, tujuan dan manfaat pembuatan Tugas Akhir serta metodologi dan sistematika pembuatan buku Tugas Akhir ini.

1.1 Latar Belakang

Semakin pesatnya perkembangan teknologi, aspek komunikasi antar komputer menjadi sangat penting. Salah satu cara komputer berkomunikasi adalah menggunakan jaringan *ad hoc*. Dalam jaringan *ad hoc* setiap komputer terkoneksi satu sama lain secara langsung tanpa bergantung pada *router* atau *access point*. Salah satu system yang menggunakan jaringan *ad hoc* adalah VANETs.

VANETs merupakan pengembangan dari *Mobile Ad Hoc Network* (MANET) yang diapkikasikan dalam kendaraan. Jaringan VANETs memungkinkan kendaraan saling berkomunikasi tanpa membutuhkan pengaturan infrastruktur tersentral ataupun server yang digunakan untuk mengontrol kerjanya. VANETs memiliki banyak *routing protocol*, salah satunya adalah *Greedy Perimeter Stateless Routing* (GPSR) yang akan digunakan pada Tugas Akhir ini.

Greedy Perimeter Stateless Protocol (GPSR) menggunakan posisi node secara *geographic* untuk mengetahui posisi *destination*. Untuk *forwarding* paketnya GPSR akan mencari node mana yang paling dekat dengan *destination* secara *greedy* tanpa tahu apakah node tersebut benar-benar bisa digunakan untuk mengirimkan paket. Oleh karena itu, performa GPSR perlu ditingkatkan dengan cara menambahkan metode *route discovery* yang akan digunakan untuk mencari rute menuju *destination.*

Pada Tugas Akhir ini, penulis akan memodifikasi *Greedy Perimeter Stateless Routing* (GPSR) dengan metode *route*

discovery protokol DSR dan menerapkan konsep *overlay network* yang diberi nama GODV. Penulis akan melakukan uji coba dan membandingkan performa protocol GPSR dan GODV dengan skenario *grid* dan skenario *real* dengan beberapa variasi node dan kecepatan.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana cara menentukan rute yang harus dilalui paket menuju *destination* dengan mengadopsi konsep *overlay network?*
2. Bagaimana pengaruh konsep *overlay network* terhadap performa GPSR?

1.3 Batasan Masalah

Permasalahan yang akan dibahas dalam Tugas Akhir ini memiliki beberapa Batasan antara lain:

1. Proses uji coba dan simulasi menggunakan *Network Simulator 2* (NS-2)
2. Pembuatan skenario simulasi VANETs menggunakan SUMO

1.4 Tujuan

Tujuan dari dibuatnya Tugas Akhir ini adalah untuk mengetahui pengaruh konsep *overlay network* terhadapa performa *routing protocol* yang telah dimodifikasi

1.5 Manfaat

Dengan dibuatnya Tugas Akhir ini adalah untuk menjadi acuan untuk topik penelitian *overlay network* pada VANETs dengan menggunakan NS-2

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Proposal Tugas Akhir menjelaskan secara garis besar tentang alur pembuatan sistem.

2. Studi literatur
   Pada tahap ini dilakukan pemahaman informasi dan literatur yang diperlukan untuk pembuatan Tugas Akhir ini. Literatur yang dipelajari dan digunakan meliputi buku referensi, artikel, jurnal, dan dokumentasi dari internet.

3. Implementasi protokol
   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini merupakan tahap mendefinisikan aplikasi yang akan diimplementasikan. Pada tahapan ini juga *prototype* sistem dibuat, yang merupakan rancangan dasar dari sistem yang akan dibuat.

4. Pengujian dan evaluasi
   Pada tahapan ini dilakukan uji coba terhadap aplikasi yang telah dibuat. Pengujian dan evaluasi dilakukan untuk mengetahui *Packet Delivery Ratio* (PDR), *End to End Delay*, dan *Routing Overhead.*

5. Penyusunan buku Tugas Akhir.
   Pada tahapan ini disusun buku yang memuat dokumentasi dari pengerjaan Tugas Akhir yang mencakup pembuatan serta hasil uji coba sistem yang telah dikerjakan.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Sistematika Penulisan untuk menyusun Buku Tugas Akhir ini adalah sebagai berikut:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, rumusan masalah, Batasan masalah, tujuan, dan manfaat serta manfaat dari pembuatan Tugas Akhir ini.

Bab II Tinjauan Pustaka

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini. Dasar teori yang digunakan adalah VANETs secara umum, GPSR, DSR, *Overlay Network*, SUMO, AWK, dan NS-2,

Bab III Perancangan

Bab ini berisi tentang pembahasan mengenai sistem yang diimplementasikan dan dilakukan uji coba.

Bab IV Implementasi

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa kode program yang digunakan untuk proses implementasi.

Bab V Uji Coba Dan Evaluasi

Bab ini berisikan tentang hasil uji coba yang telah dilakukan dengan menghitung *Packet Delivery Ratio* (PDR), *End to End Delay*, dan *Routing Overhead.*

Bab VI Kesimpulan Dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan serta masalah-masalah yang dialami pada saat mengerjakan Tugas Akhir dan saran untuk pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA

Pada bab ini akan dibahas mengenai teori-teori yang berkaitan dengan pembuatan Tugas Akhir ini. Penjelasan dilakukan dengan tujuan untuk memberikan gambaran secara umum terhadap *routing protocol* serta definisi yang digunakan dalam pembuatan Tugas Akhir.

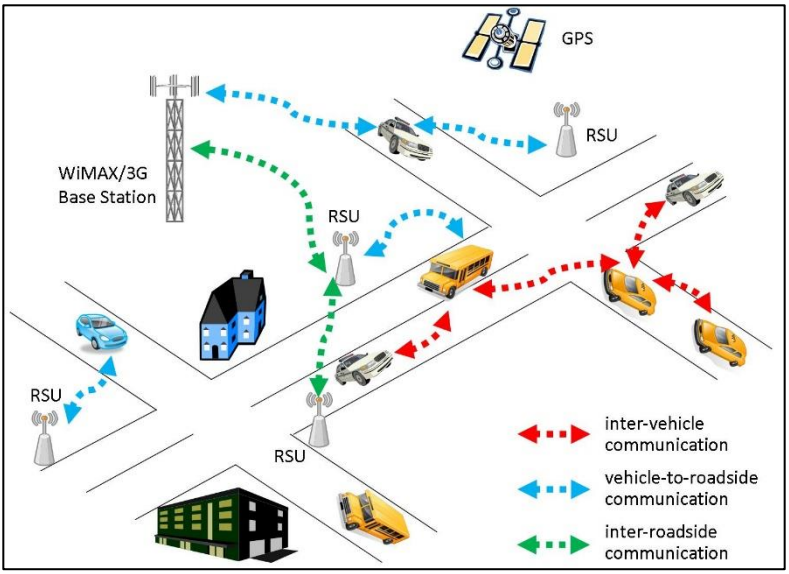
2.1 *Vehicular Ad hoc Network* (VANETs)

Vehicular Ad hoc Network merupakan bentuk dari jaringan nirkabel yang bersifat *ad hoc* dan merupakan pengembangan dari *Mobile Ad hoc Network* (MANET) yang mempertimbangkan semua kendaraan di dalam jaringan sebagai node yang digunakan untuk berkomunikasi dengan kendaraan lainnya pada cakupan tertentu.

Pada MANET maupun VANETs, node yang bergerak bergantung pada *ad hoc routing protocol* untuk menentukan bagaimana cara mengirimkan pesan dari *source node* menuju *destination node*. *Ad hoc routing protocol* dapat diklasifikasikan menjadi dua kategori, yaitu *Topology-Based Routing* dan *Geographic Position-Based Routing*. Meskipun menggunakan *routing protocol* yang sama, VANETs memiliki perbedaan karakteristik dengan MANET. Pergerakan pada VANETs dibatasi oleh bentuk jalan yang dilalui kendaraan dan kecepatan kendaraan yang cenderung lebih cepat.

Terdapat dua jenis node yang tergabung pada VANETs, yaitu *Road-side Unit* (RSU) dan *On-board Unit* (OBU). RSU merupakan node yang terpasang pada bagian jalan yang terhubung dengan jaringan *backbone* untuk memberikan informasi-informasi penting kepada OBU. Informasi yang dapat diberikan oleh RSU adalah batas kecepatan, status lampu lalu lintas, dan lain-lain. OBU merupakan kendaraan yang berjalan pada suatu bidang jalan.

Informasi yang dapat diberikan oleh OBU antara lain adalah kecepatan, lampu sen, status rem, sudut kemudi, dan kondisi lalu lintas. Ada dua tipe komunikasi yang ada dalam VANETs, yaitu antara kendaraan (yang memiliki OBU) dengan kendaraan yang lain dan antara kendaraan dengan RSU. Ilustrasi mengenai komunikasi dalam VANETs dapat dilihat pada Gambar 2.1



Gambar 2.1 Ilustrasi komunikasi dalam VANETs **[1]**

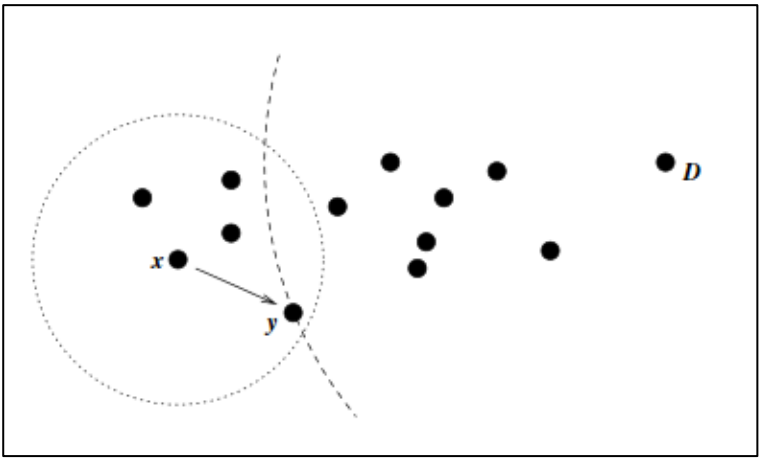
2.2 *Greedy Perimeter Stateless Routing* (GPSR)

Greedy Perimeter Stateless Routing merupakan *Geographical Routing Protocol* yang menggunakan *position-based routing*, di mana setiap node saling mengetahui posisi sendiri dan posisi tetangga terdekatnya. *Routing protocol* GPSR akan mengirimkan *hello message* secara berkala untuk memperbarui

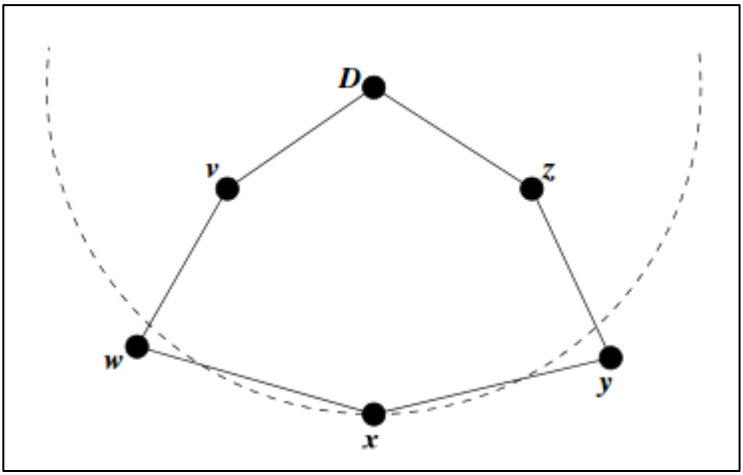
informasi lokasi geografis node yang msaih berada pada jangkauan transmisi.

Protokol GPSR menggunakan dua metode untuk melakukan pengiriman paket data, yaitu:

- *Greedy forwarding*
  Greedy forwarding merupakan metode utama yang digunakan untuk pengiriman paket data. *Greedy forwarding* akan meneruskan paket data kepada node yang paling dekat dengan *destination node*. Gambaran tentang metode *greedy forwarding* dapat dilihat pada Gambar 2.2.
- *Perimeter forwarding*
  Perimeter forwarding digunakan apabila pada area tertentu ketida tidak ada node yang lebih dekat dengan *destination node*. *Perimeter forwarding* akan bekerja apabila *greedy forwarding* tidak bisa menemukan node yang paling dekat dengan *destination node*. Gambaran tentang *perimeter forwarding* dapat dilihat pada Gambar 2.3. *[2]*



Gambar 2.2 Metode *greedy forwarding* **[2]**

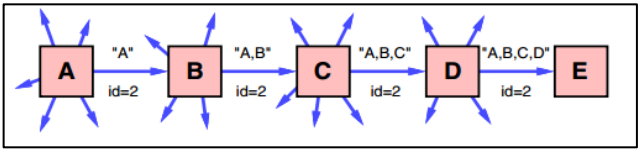


Gambar 2.3 Metode *perimeter forwarding* **[2]**

2.3 *Dynamic Source Routing* (DSR)

Dynamic Source Routing merupakan *routing protocol* yang bersifat *reactive* yang artinya protokol ini akan melakukan *route discovery* jika sebuah node inisiator ingin berkomunikasi dengan node yang lain, oleh karena itu *network traffic* dapat diturunkan. Sebelum melakukan pengiriman paket data, DSR akan melakukan *route discovery* kemudian menyimpan *route path* pada *header* paket yang digunakan untuk pengiriman paket data. Pada DSR, *intermediate node* tidak perlu mengurus informasi *routing*. Proses *route discovery* di mulai saat sebuah node *source* ingin berkomunikasi dengan node *destination*. Node *source* akan mengirim *route request* (RREQ) menuju node *destination*. Node *intermediate* akan menyisipkan dirinya pada *route path* yang ada pada *header* RREQ jika belum pernah menerima RREQ tersebut dan menyimpan nomor RREQ pada *routing table,* lalu node tersebut akan melakukan *broadcast* ulang ke node yang lain. Apabila pada *routing table* node sudah ada nomor RREQ tersebut

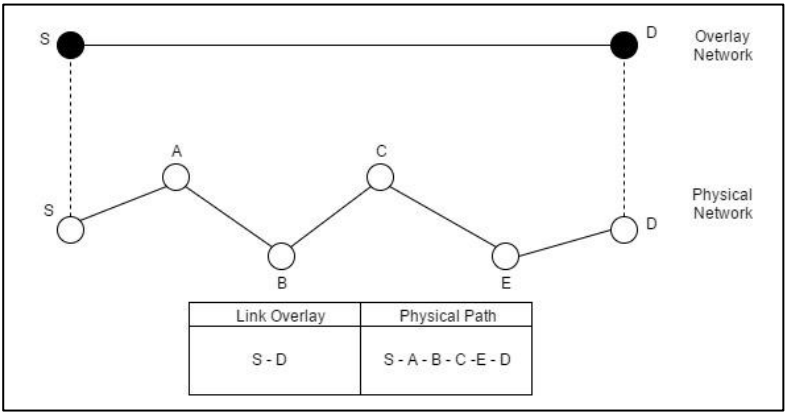
maka paket RREQ akan di drop. Mekanisme *route discovery* pada DSR dapat dilihat pada Gambar 2.4. [3]



Gambar 2.4 Mekanisme *route discovery* pada DSR **[3]**

2.4 *Overlay Network*

Overlay Network adalah suatu jaringan yang dibangun di atas jaringan lain, yang artinya setiap node saling terhubung melalui jaringan *virtual* di atas jaringan fisik. Pada tugas akhir ini konsep *overlay network* di gunakan pada proses pengiriman paket data. Node *source* secara langsung mengirimkan paket data menuju node *destination* pada *overlay network*, namun jika lihat dari *physical network* pengiriman paket data melalui node-node. Gambaran tentang *overlay network* dapat dilihat pada Gambar 2.5. [4]



Gambar 2.5 Konsep *overlay network*

2.5 *Simulation of Urban Mobility* (SUMO)

Simulation of Urban Mobility merupakan sebuah aplikasi simulasi lalu lintas jalan yang di desain untuk menangani jaringan jalan yang besar. SUMO memiliki lisensi dibawah GPL yang dikembangan sejak tahun 2000 dengan tujuan untuk mengakomodasi penelitian-penelitian yang melibatkan kendaraan di jalan raya, terutama daerah dengan penduduk padat. [5]

SUMO terdiri dari beberapa *tools* yang dapat membantu pembuatan simulasi lalu lintas. *Tools* dari SUMO yang digunakan pada Tugas Akhir ini adalah sebagi berikut:

- *netgenerate*
  netgenerate merupakan *tools* yang berfungsi untuk membuat peta seperti *grid*, *spider*, dan *random network*. *Netgenerate* juga berfungsi untuk menentukan kecepatan maksimum jalan dan membuat *traffic light* pada peta. Hasil dari *netgenerate* berupa *file* dengan ekstensi *.net.xml. Pada Tugas Akhir ini, *netgenerate* digunakan untuk membuat peta pada simulasi skenario *grid*.
- *netconvert*
  netconvert merupakaan *tools* yang berfungsi untuk melakukan impor peta jalan dari sumber lain dan menghasilkan peta jalan. Pada Tugas Akhir ini, *netconvert* digunakan untuk mengkonversi peta dari *OpenStreetMap*.
- *randomTrips.py*
  randomTrips.py merupakan *tools* yang digunakan untuk membuat rute perjalanan kendaraan secara acak dalam peta simulasi yang telah dibuat.
- *Duarouter*
  Duarouter merupakan *tools* yang berfungsi untuk membuat detail perjalanan setiap kendaraan

berdasarkan *output* dari *randomTrips.py* dan *netgenerate* ataupun *netconvert*.

- *Sumo*
  Sumo merupakan program untuk melakukan simulasi lalu lintas berdasarkan data-data yang didapatkan dari hasil *netgenerate* atau *netconvert* dan *duarouter*.
- *Sumo-gui*
  Sumo-gui merupakan aplikasi GUI untuk melihat simulasi yang dilakukan SUMO.
- *traceExporter.py*
  traceExporter.py merupakan *tools* yang bertujuan untuk mengkonversi output dari *sumo* menjadi format yang dapat digunakan oleh NS-2. Pada Tugas Akhir ini *traceExporter.py* digunakan untuk mengkonversi data menjadi *file* dengan ekstensi *.TCL.

2.6 *OpenStreetMap*

OpenStreetMap merupakan sebuah proyek kolaboratif untuk membuat sebuah peta dunia yang dapat dengan bebas diubah oleh siapapun. Dua buah faktor pendukung dalam pembuatan dan perkembangan OSM adalah kurangnya ketersediaan dari informasi peta mengenai sebagian besar daerah di dunia dan munculnya alat navigasi portable yang terjangkau. OSM dianggap sebagai contoh utama dalam informasi geografis yang diberikan secara sukarela.

Sejak diluncurkan hingga sekarang OSM telah berkembang hingga memiliki lebih dari dua juta pengguna yang terdaftar yang dapat mengambil data menggunakan survei manual, piranti GPS, *aerial photography*, dan sumber bebas lainnya. Data yang terdapat pada OSM berada dalam lisensi *Open Database License* sehingga data dari OSM dapat dengan bebas digunakan oleh semua orang.

Pada Tugas Akhir ini penulis menggunakan data yang tersedia pada OSM untuk membuat simulasi skenaio *real* berdasarkan peta Kota Surabaya. [6]

2.7 JOSM

JOSM (*Java OpenStreetMap Editor)* adalah sebuah alat untuk menyunting data yang didapatkan dari *OpenStreetMap.* Aplikasi JOSM dapat diunduh pada alamat http://josm.openstreetmap.de/. Penulis menggunakan aplikasi ini untuk merapikan potongan peta yang diunduh dari *OpenStreetMap* yang digunakan untuk simulasi skenario *real.* [7]

2.8 AWK

AWK merupakan suatu Bahasa Pemrograman yang digunakan *text processing* dan biasanya digunakan untuk ekstraksi data dan *reporting.* AWK bersifat *data-driven* yang berisikan kumpulan perintah yang akan dijalankan pada data tekstural secara langsung pada *file* atau digunakan sebagai bagian dari *pipeline*. Pada Tugas Akhir ini penulis menggunakan AWK untuk memproses data yang dihasilkan dari simulasi NS-2 untuk menganalisis *packet delivery ratio*, *end to end delay,* dan *routing overhead*. [8]

2.9 *Network Simulator 2* (NS-2)

Network Simulator 2 (NS-2) merupakan sebuah *descrete event simulator* yang gunakan untuk membanti penelitian pada bidang jaringan komputer. Pengembangan NS dimulai pada tahun 1989 sebagai sebuah varian dari REAL *network simulator*. Pada tahun 1995, pengembangan NS didukung oleh DARPA melalui *project* di LBL, XeroX PARC, UCB, dan USC/ISI. NS kemudian memasuki versi 2 pada tanggal 31 Juli 1995. Sekarang, pengembangan NS didukung oleh DARPA melalui SAMAN dan NSF melalui CONSER beserta peneliti lainnya.

Terdapat dua buah *major version* dari NS yang masih dikembangkan, yaitu NS-2 dan NS-3. Pada Tugas Akhir ini penulis menggunakan NS-2 dengan versi terbaru (ns-2.35) yang dirilis pada tanggal 14 November 2011. Dalam membuat sebuah simulasi

jaringan komputer, NS-2 menggunakan dua buah Bahasa pemrograman, yaitu C++ dan TCL yang dapat digunakan pada sistem operasi Linux, FreeBSD, OS X, Solaris, dan Windows. [9]

2.9.1 Instalasi NS-2

Instalasi NS-2 dilakukan pada sistem operasi Ubuntu 14.04. Proses instalasi NS-2 dapat dilihat pada lampiran A.11.

2.9.2 *Patching* Protokol GPSR

Karena pada NS-2 versi 2.35 protokol GPSR tidak ada, maka perlu dilakukan *patching* terhadap protokol GPSR. Proses *patching* protokol GPSR dapat dilihat pada lampiran A.12.

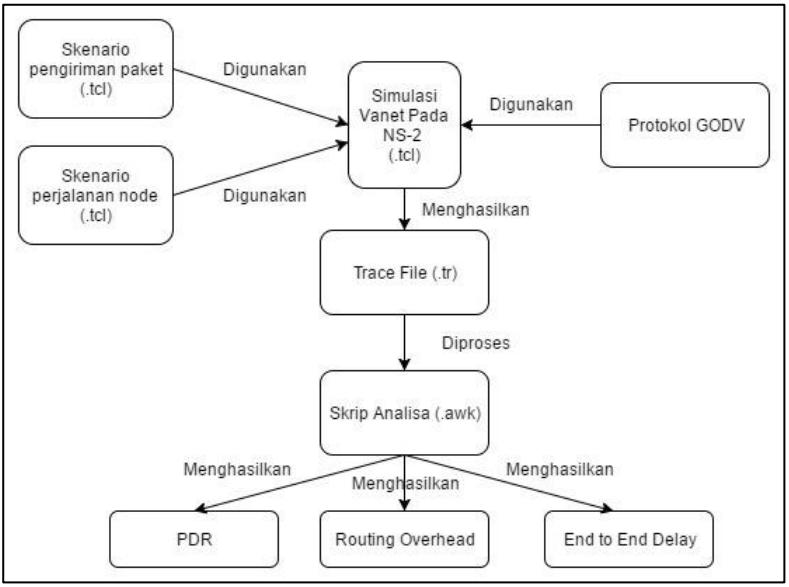
[Halaman ini sengaja dikosongkan]

BAB III
PERANCANGAN

Pada bab ini akan dijelaskan mengenai perancangan sistem yang dibuat dalam Tugas Akhir ini. Mulai dari deskripsi umum, perancangan skenario, hingga perancangan protokol yang dibuat.

3.1 Deskripsi Umum

Pada Tugas Akhir ini penulis mengimplementasikan kombinasi protokol GPSR dan DSR (GODV) dengan menerapkan konsep *overlay network* yang dijalankan pada simulator NS-2. Diagram rancangan simulasi dapat dilihat pada Gambar 3.1.

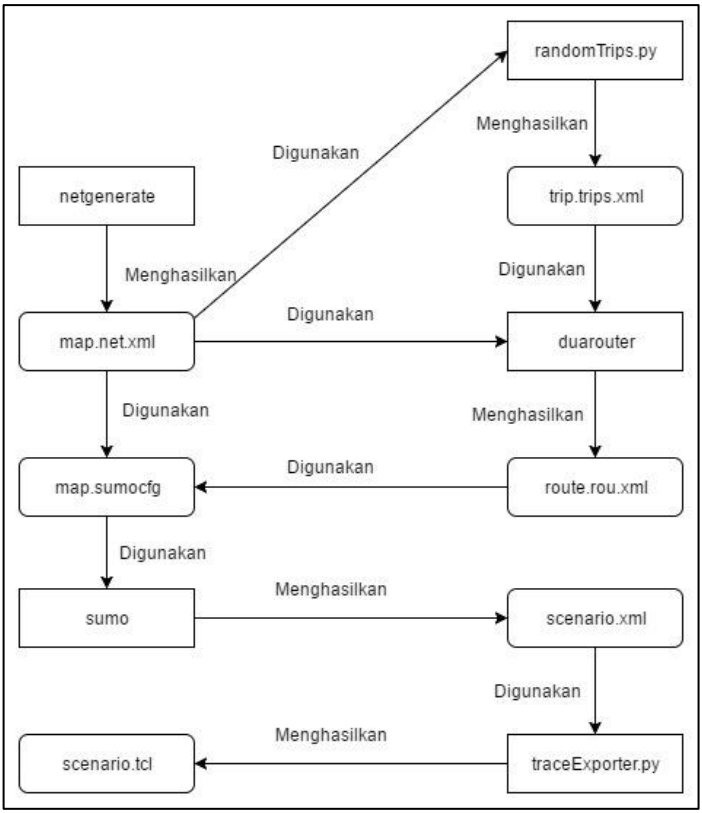


Gambar 3.1 Diagram rancangan simulasi.

Dalam Tugas Akhir ini skenario yang digunakan adalah peta berbentuk *grid*. Peta berbetuk *grid* dibuat menggunakan *tools* dari SUMO. Hasil simulasi dari SUMO yang berupa pergerakan node-node akan digunakan pada simulator NS-2. Protokol pengiriman data yang digunakan untuk melakukan simulasi VANETs adalah GPSR dan GODV. Hasil simulasi VANETs yang berupa *trace file* kemudian dianalisis menggunakan skrip AWK untuk mendapatkan *packet delivery ratio*, *end to end delay*, dan *routing overhead*. Hasil analisis ini digunakan untuk mengukur tingkat reliabilitas pengiriman data antara protokol GPSR dengan GODV.

3.2 Perancangan Skenario *Grid*

Perancangan skenario *grid* diawali dengan menentukan panjang dan lebar peta *grid* yang akan digunakan. Selanjutnya tentukan berapa persimpangan yang digunakan dan berapa jarak antar persimpangan. Sebagai contoh, peta *grid* dengan luas 800 x 800 membutuhkan 9 buah persimpangan dan jarak antar persimpangan adalah 100. Peta *grid* dibuat menggunakan *tools* dari SUMO, yaitu *netgenerate*. Selain pengaturan banyak persimpangan dan jarak antar persimpang, pada *netgeneate* pula kecepatan maksimum kendaraan diatur. Hasil keluaran *netgenerate* adalah *file* dengan ekstensi **.net.xml*. Peta ini digunakan untuk membuat *file* deskripsi pergerakan kendaraan menggunakan *tools randomTrips.py*. Hasil keluaran *randomTrips.py* adalah *file* dengan ekstensi **.trips.xml*. Hasil dari *netgenerate* dan *randomTrips.py* digunakan oleh *duarouter*. SUMO akan menggunakan hasil keluaran *netgenerate* dan *duarouter* untuk menghasilkan *file* dengan ekstensi **.xml*. Hasil keluaran dari SUMO tadi digunakan oleh *tools* dari SUMO yaitu *traceExporter.py* untuk dijadikan *file* dengan ekstensi **.tcl*. Untuk lebih jelasnya tentang mekanisme perancangan skenario *grid* dapat dilihat pada Gambar 3.2.

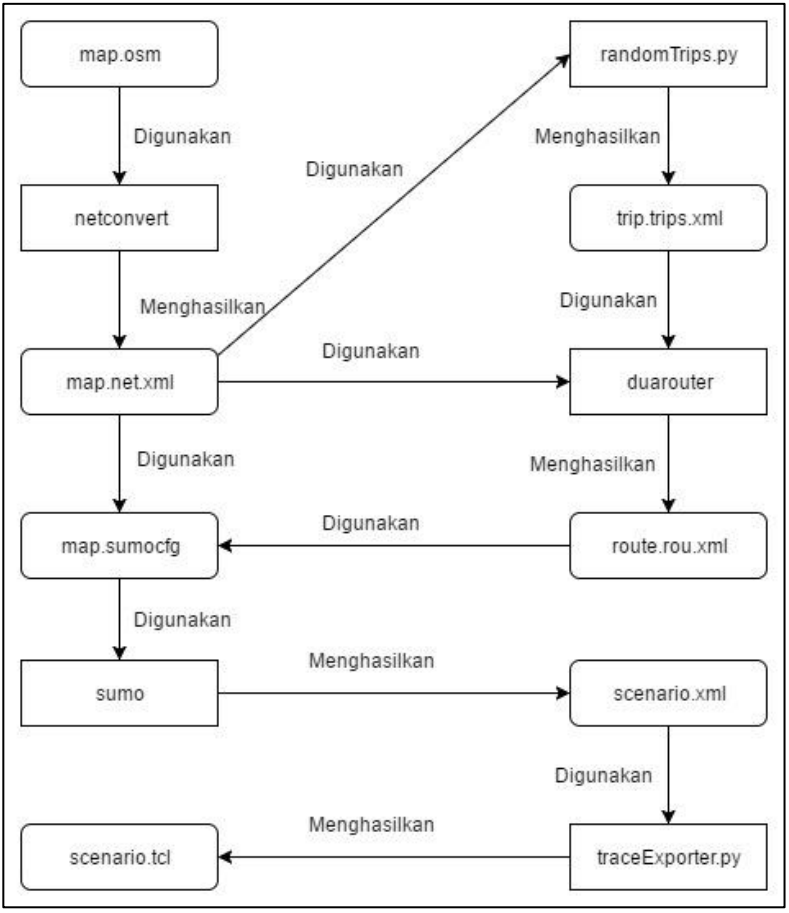


Gambar 3.2 Mekanisme pembuatan skenario *grid*

3.3 Perancangan Skenario *Real*

Perancangan skenario *real* dimulai dengan memilih daerah yang akan digunakan untuk simulasi. Pada *OpenStreetMap* lakukan *export* untuk mengunduh daerah yang sudah dipilih. Dengan bantuan JOSM, rapikan peta yang sudah di unduh dari *OpenStreetMap*. Peta yang sudah dirapikan nantinya akan di konversikan menjadi *file* dengan ekstensi *.net.xml menggunakan

tools dari SUMO yaitu *netconvert*. Hasil dari *netconvert* akan digunakan *randomTrips.py* dan *route2trips* untuk membuat *file* deskripsi perjalanan secara acak. Mekanisme pembuatan skenario *real* secara lengkap dapat dilihat pada Gambar 3.3.



Gambar 3.3 Mekanisme pembuatan skenari *real*

3.4 Perancangan Simulasi pada NS-2

Simulasi VANETs pada NS-2 dilakukan dengan menggunakan skenario yang hasilkan oleh SUMO dan digabungkan dengan skrip TCL yang berisikan konfigurasi mengenai lingkungan simulasi.

3.5 Perancangan Metrik Analisis

Berikut ini adalah parameter-parameter yang dianalisis dalam Tugas Akhir ini:

3.5.1 *Packet Delivery Ratio* (PDR)

Packet delivery ratio [10] adalah prosentasi antara jumlah paket yang dikirimkan dan paket yang diterima. Rumus untuk menghitung PDR adalah sebagai berikut:

$$PDR = \frac{received}{sent} \times 100\%$$

Keterangan:

PDR = *packet delivery ratio*

Received = banyak paket data yang diterima

Sent = banyak paket data yang dikirimkan

3.5.2 *End to End Delay*

End to end delay [10] adalah waktu jeda antara waktu paket data dikirimkan dengan waktu paket data diterima. Rumus untuk menghitung *end to end delay* adalah sebagai berikut:

$$E2E = \frac{\sum_{i=0}^{sent} t_{received[i]} - t_{sent[i]}}{sent}$$

Keterangan:

E2E = *end to end delay*

$t_{received[i]}$ = waktu pengiriman paket data (detik)

$t_{sent[i]}$ = waktu pengiriman paket data (detik)

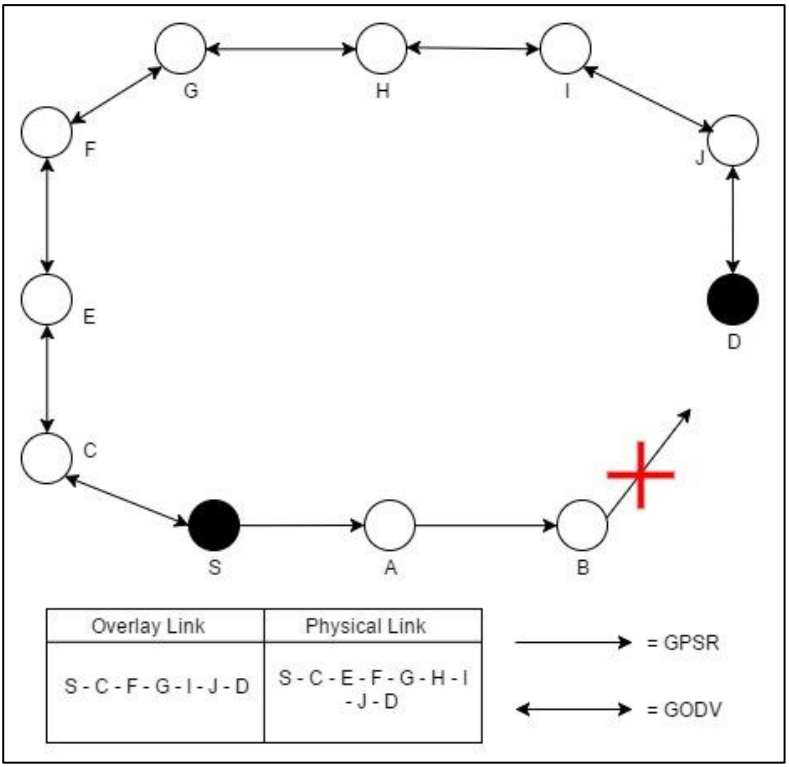
sent = banyaknya paket data yang dikirimkan

3.5.3 *Routing Overhead*

Routing overhead [10] adalah jumlah paket *routing* yang diperlukan untuk komunikasi di dalam sebuah jaringan. *Routing overhead* dihitung berdasarkan jumlah paket *routing* dibagi dengan jumlah paket data.

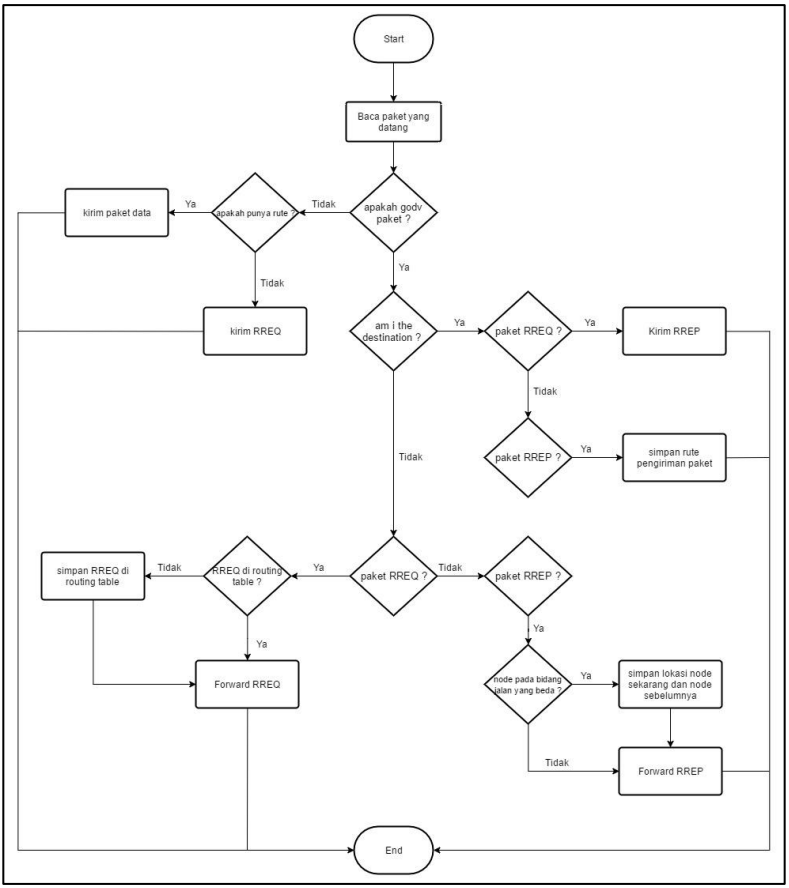
3.6 Perancangan Protokol GODV

Protokol GODV merupakan modifikasi dari protokol GPSR yang menambahkan metode *route discovery* dari protokol DSR. Modifakasi dilakukan untuk mencari rute untuk melakukan pengiriman paket. Ilustrasi tentang protokol GODV dapat dilihat pada Gambar 3.4.



Gambar 3.4 Ilustrasi protokol GODV

Protokol GODV menggunakan rute yang didapatkan oleh proses *route discovery* karena pada proses *route discovery* terdapat sebuah paket yang berhasil dikirimkan kepada node *destination*. Rute yang digunakan oleh proses *route discovery* kemudian digunakan oleh node *source* untuk mengirimkan paket data menuju node *destination*. *Flowchart* yang menggambarkan alur proses pengiriman data dari proses *route discovery* hingga pengiriman paket data dapat dilihat pada Gambar 3.5.



Gambar 3.5 *Flowchart* protokol GODV

BAB IV
IMPLEMENTASI

Bab ini membahas mengenai implementasi dari perancangan sistem yang telah dijelaskan pada bab sebelumnya.

4.1 Lingkungan Implementasi Protokol

Lingkungan implementasi protokol rinciannya dapat dilihat pada Tabel 4.1.

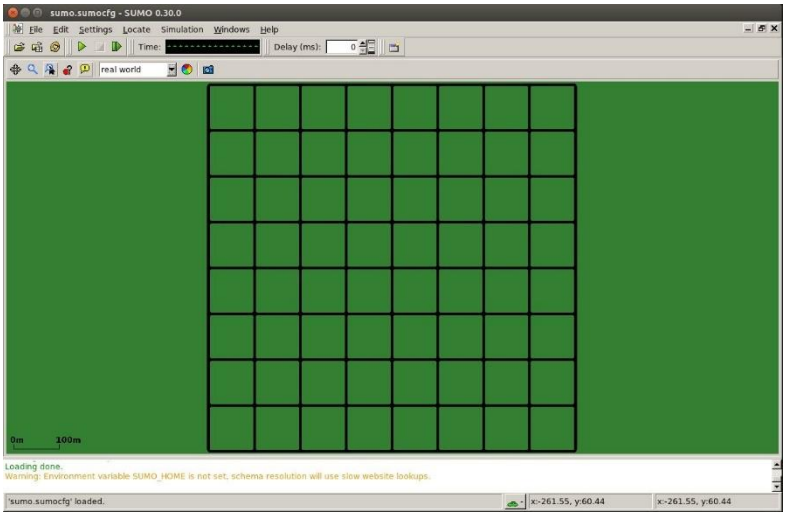
Tabel 4.1 Spesifikasi lingkungan implementasi protokol

Komponen	Spesifikasi
CPU	Inter core i7 @2.60GHz
Memori	4GB
Penyimpanan	40GB
Sistem Operasi	Ubuntu 14.04 LTS 64bit
Network Simulator	2.35
SUMO	0.30

4.2 Implementasi Skenario *Grid*

Skenario *grid* dibuat menggunakan *tools* yang telah disediakan oleh SUMO, yaitu *netgenerate*. Untuk membuat peta dengan luas 800 x 800 meter, dengan panjang antar persimpangan 100 meter dibutuhkan 9 titik persimpangan. Untuk kecepatannya bisa di variasikan sesuai kebutuhan skenario. Sebagai contoh dengan kecepatan 10m/s. Perintah untuk membuat skenario grid adalah sebagai berikut:

```
netgenerate --grid --grid.number=9 --grid.length=100 --
default.speed=10 --tls.guess=1 --output-file=map.net.xml
```



Gambar 4.1 Peta skenario *grid* 800m x 800m

Gambar 4.1 merupakan hasil dari perintah *netgenerate*. Namun pada peta skenario di atas belum terdapat node-node, sehingga dibutuhkan *tools randomTrips.py* yang sudah disediakan juga oleh SUMO. Perintah untuk menggunakan *randomTrips.py* adlaah sebagai berikut:

randomTrips.py -n map.net.xml -e 100 -l --trip-attributes="departLane=\"best\" departSpeed=\"max\" departPos=\"random_free\"" --o trip.trips.xml

Perintah di atas akan membuat node sebanyak seratus buah. *File* keluaran dari *randomTrips.py* nantinya akan digunakan oleh *tools* SUMO yang lain untuk membuat rute perjalanannya. *Tools* yang digunakan adalah *duarouter*. Perintah untuk menggunakan *duarouter* adalah sebagai berikut:

duarouter -n map.net.xml -t trip.trips.xml -o route.rou.xml --ignore-errors --repair

Hasil dari *duarouter* nantinya akan digunakan SUMO untuk membuat skenario perjalanan node. SUMO membutuhkan *file* hasil dari *netgenerate* dan *duarouter*. Perintah untuk menggunakan SUMO adalah sebagai berikut:

sumo -b 0 -e 200 -n map.net.xml -r route.rou.xml --fcd-output=scenario.xml

Hasil dari SUMO selanjutnya akan dikonversi menjadi *file* dengan ekstensi *.tcl agar bisa digunakan oleh NS-2. *Tools* yang digunakan untuk melakukan konversi adalah *traceExporter.py*. Perintah untuk menggunakan *traceExporter.py* adalah sebagai berikut:

traceExporter.py --fcd-input=scenario.xml --ns2mobility-output=scenario.tcl

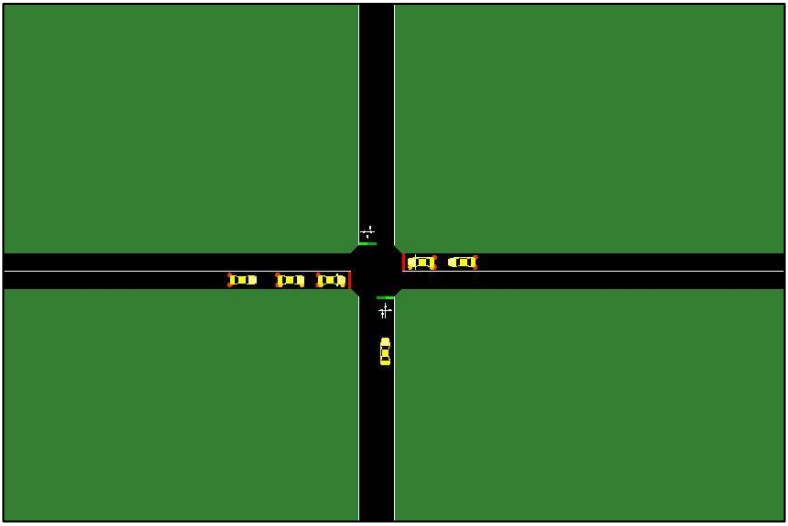
Pada protokol GPSR Ke Liu yang digunakan pada Tugas Akhir ini, semua node perlu didefinisikan pada detik ke-0 sebelum node melakukan pergerakan. Oleh karena itu perlu adanya modifikasi pada *randomTrips.py* untuk memudahkan dalam pembuatan skenario. Gambar 4.2 merupakan potongan kode yang ada dalam *randomTrips.py* sebelum di modifikasi. Bagian yang harus dimodifikasi adalah bagian *depart* yang harus mengeluarkan nilai 0. Untuk itu *randomTrips.py* perlu dimodifikasi yang hasil modifikasinya dapat dilihat pada Gambar 4.3. Cuplikan skenario *grid* pada *sumo-gui* dapat dilihat pada

```
fouttrips.write('<trip id="%s" depart="%.2f" from="%s" to="%s"%s%s/>\n' % (
   label, depart, source_edge.getID(), sink_edge.getID(), via, options.tripattrs))
```

Gambar 4.2 *randomTrips.py* sebelum modifikasi

```
fouttrips.write('<trip id="%s" depart="%.2f" from="%s" to="%s"%s%s/>\n' % (
   label, 0, source_edge.getID(), sink_edge.getID(), via, options.tripattrs))
```

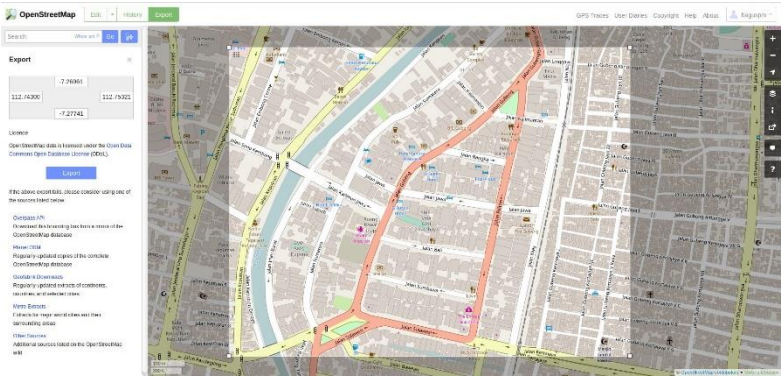
Gambar 4.3 *randomTrips.py* setelah modifikasi



Gambar 4.4 Cuplikan skenario *grid* pada *sumo-gui*

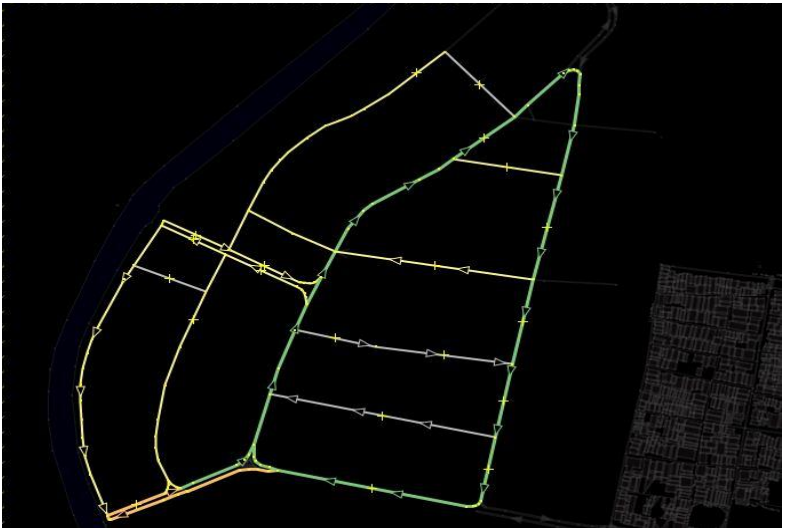
4.3 Implementasi Skenario *Real*

Skenario *real* ini menggunakan peta Kota Surabaya yang didapatkan menggunakan *OpenStreetMap* dengan cara melakukan ***export*** pada daerah yang ingin digunakan. Pada Tugas Akhir ini daerah yang digunakan adalah di sekitar Gubeng, Surabaya seperti yang ditampilkan pada Gambar 4.5.



Gambar 4.5 Peta Surabaya daerah Gubeng

Peta yang diambil dari *OpenStreetMap* perlu dimodifikasi untuk menghilangkan gedung-gedung yang ada dan memperbaiki jalan-jalan yang terputus. Aplikasi yang digunakan untuk memodifikasi adalah JOSM. Hasil modifikasi dari JOSM dapat dilihat pada Gambar 4.6.

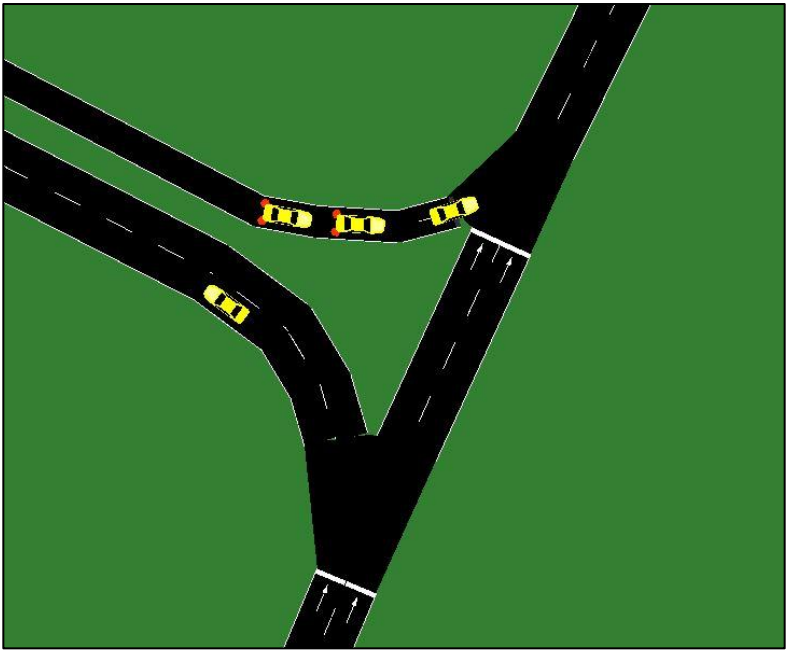


Gambar 4.6 Hasil modifikasi menggunakan JOSM

Setelah melakukan modifikasi, berarti peta sudah siap di konversi menjadi *file* dengan ekstensi *.net.xml. *Tools* yang digunakan untuk proses konversi adalah *netconvert*. Perintah untuk melakukan proses konversi adalah sebagai berikut:

netconvert –osm-files map.osm –output-file map.net.xml

Hasil dari *netconvert* nantinya akan digunakan oleh *tools* dari SUMO yang lain untuk dijadikan bentuk *file* dengan ekstensi *.TCL. Mulai dari sini, proses pembuatan skenario *real* sama dengan proses pembuatan skenario *grid*. Cumplikan *sumo-gui* untuk skenario *real* dapat dilihat pada Gambar 4.7.



Gambar 4.7 Cuplikan skenario *real* pada *sumo-gui*

4.4 Implementasi Protokol GODV

Protokol GODV merupakan modifikasi dari protokol GPSR yang menerapkan metode *route discovery* milik protokol DSR. Modifikasi yang dilakukan pada protokol GODV adalah sebagai berikut:

- Implementasi *Header* protokol DSR pada *Header* GPSR
- Implementasi *Routing Table* pada GPSR
- Modifikasi *Class* GPSRAgent
- Implementasi *Route Discovery* pada GPSR
- Implementasi konsep *overlay network* pada RREP
- Modifikasi *forwarding node* pada GPSR

Data-data yang dimodifikasi untuk implementasi protokol GODV adalah **gpsr.cc**, **gpsr.h**, **gpsr_packet.h**, **godv_rtable.h**, dan **godv_rtable.cc**. Data-data tersebut dapat ditemukan pada direktori **ns-2.35/gpsr**.

4.4.1 Implementasi *Header* Protokol DSR pada *Header* GPSR

Implementasi *header* protokol DSR pada *header* GPSR dengan cara memodifiaksi *file* **gpsr_packet.h**. Pada *file* tersebut berisi *header-header* yang digunakan oleh GPSR. Penulis membuat sebuah *struct* baru dengan nama **hdr_godv_data** yang berisi informasi RREQ, RREP, *node path*, *overlay node*, dan *overlay node position*.

Struct hdr_godv_data memilik satu variabel dengan tipe data *u_int8_t*, sembilan variabel dengan tipe data *integer*, dua variabel dengan tipe data *nsaddr_t*, dan satu variabel dengan tipe data *double*. *Node path* dan *overlay node* didefinisikan sebagai *array of nsaddr_t*, pada *node path* berisi rute yang dilalui oleh RREQ, pada *overlay node* berisi rute yang harus dilalui saat pengiriman paket yang digunakan oleh node *source*. *Overlay node position* didefinisikan sebagai *array of double* yang berisi posisi-

posisi node yang ada pada *overlay node*. Selanjutnya perlu dilakukan modifikasi pada **union hdr_all_gpsr** supaya *struct* hdr_godv_data dapat terbaca dengan cara menambahkan *struct* hdr_godv_data pada union hdr_all_gpsr.

```
union hdr_all_gpsr {
  hdr_gpsr              gh;
  hdr_gpsr_hello        ghh;
  hdr_gpsr_query        gqh;
  hdr_gpsr_data         gdh;
  hdr_godv_data         godv;
};
```

Gambar 4.8 Modifikasi union hdr_all_gpsr

Modifikasi union hdr_all_gpsr dapat dilihat pada Gambar 4.8. hdr_godv_data ditambahkan dalam union hdr_all_gpsr. Untuk implementasi *struct* hdr_godv_data dapat dilihat pada Gambar 4.10. Kemudian penulis melakukan pendefinisian beberapa variabel supaya hdr_godv_data dapat digunakan. **HDR_GODV_DATA(p)** digunakan untuk mengakses *struct* hdr_godv_data. **GPSRTYPE_ROUTEDISC** digunakan untuk membuat paket jenis baru pada protokol GODV yang dibuat. Untuk lebih jelasnya mengenai pendefinisian yang dilakukan penulis dapat dilihat pada Gambar 4.9.

```
#define GPSRTYPE_ROUTEDISC 0x03 //Route Discovery
#define HDR_GODV_DATA(p) ((struct
hdr_godv_data*)hdr_gpsr::access(p))
```

Gambar 4.9 Pendefinisian variabel pada gpsr_packet.h

```
struct hdr_godv_data {
  u_int8_t type_;
  int pValid ; //packet godv valid
  //RREQ PACKET
  int reqValid; //is RREQ
  int rq_id; //Request ID
  int rq_ttl;
  //RREP PACKET
  int repValid; //is RREP
  int rp_id; //Reply ID
  nsaddr_t nodePath[16]; //Path of node
  nsaddr_t overlayNode[16]; //list of Overlay Node;
  double path[16][2]; //loc X,Y node
  int pathLen; //length path
  int length; //length nodePath
  int index;
  inline int size(){
    int sz =
      9*sizeof(int)+
      (16*2+1)*sizeof(u_int8_t) +
      16*2*sizeof(double) ;
    return sz;
  }
};
```

Gambar 4.10 Implementasi *struct* hdr_godv_data

4.4.2 Implementasi *Routing Table* pada GPSR

Implementasi *routing table* digunakan untuk mengurangi *routing overhead* dan sebagai acuan untuk sebuah node apakah telah menerima paket yang sama. *File-file* yang diimplementasikan adalah **godv_rtable.h**, **godv_rtable.cc**, dan **gpsr.h**. *Routing table* bersifat lokal, maksudnya setiap node memiliki *routing table*

sendiri sehingga node satu dengan yang lain memiliki *routing table* yang berbeda. *Routing table* berisi daftar node-node yang melakukan RREQ. Implementasi *routing table* pada GPSR dapat dilihat pada Gambar 4.11.

```
#ifndef __godv_rtable_h__
#define __godv_rtable_h__
#include <config.h>
struct godv_rt_entry{
        nsaddr_t rt_id;
        int rt_reqno;
};
class godv_rtable{
        struct godv_rt_entry *rt;
        int len;
public:
        godv_rtable();
        int rt_lookup(nsaddr_t id);
        int rt_getrid(nsaddr_t id);
        void rt_add(nsaddr_t id, int reqno);
};
#endif
```

Gambar 4.11 Implementasi godv_rtable.h

Ketika sebuah node menerima paket RREQ maka node tersebut akan mengecek pada *routing table* yang dimilikinya. Apabila paket RREQ sudah pernah diterima node tersebut, maka paket RREQ tersebut akan di drop. Apabila paket RREQ belum pernah diterima node tersebut, maka node tersebut akan menyimpan dari mana RREQ ini dikirim dan *request id* paket RREQ tersebut. Selanjutnya menambahkan **#include godv_rtable.h** pada gpsr.h supaya *file* godv_rtable.h dan godv_rtable.cc dapat diakses dari gpsr.h.

4.4.3 Modifikasi *Class* GPSRAgent

Modifikasi *Class* GPSR dilakukan pada *file* **gpsr.h** dan **gpsr.cc**. Pada *file* gpsr.h, penulis melakukan pendefinisian beberapa variabel yang digunakan untuk *route discovery*. Variabel **rt** digunakan untuk mengakses *routing table*, variabel **rq_id** merupakan nomor request yang telah dilakukan node tersebut, variabel **valid_route** merupakan *identifier* apakah node tersebut sudah mempunya rute perjalanan menuju *destination*, variabel **src** dan **dst** digunakan untuk menyimpan node *source* dan node *destination*, variabel **routeLength** menyimpan panjang rute yang harus ditempuh paket data menuju *destination*, variabel **thePath** menyimpan posisi-posisi yang harus dilewati paket data menuju *destination* akhir. Untuk lebih jelasnya mengenai pendefinisian pada gpsr.h dapat dilihat pada Gambar 4.12.

```
godv_rtable *rt;
int rq_id;
bool valid_route;
nsaddr_t src;
nsaddr_t dst;
double thePath[16][2];
int routeLength;
```

Gambar 4.12 Pendefinisian variabel pada gpsr.h

Selanjutnya variabel-variabel yang telah didefinisikan seperti pada Gambar 4.12 dipanggil pada *file* gpsr.cc. Variabel-variabel tersebut diinisiasi pada *class* GPSRAgent. Gambar 4.13 menunjukkan *class* GPSRAgent pada *file* gpsr.cc beserta variabel-variabel yang ada pada Gambar 4.12.

```
GPSRAgent::GPSRAgent() : Agent(PT_GPSR),
                         hello_timer_(this), query_timer_(this),
                         my_id_(-1), my_x_(0.0), my_y_(0.0),
                         recv_counter_(0), query_counter_(0),
                         query_period_(INFINITE_DELAY)
{
  bind("planar_type_", &planar_type_);
  bind("hello_period_", &hello_period_);

  sink_list_ = new Sinks();
  nblist_ = new GPSRNeighbors();
  rt = new godv_rtable();
  src = NULL;
  dst = NULL;
  rq_id = 1;
  valid_route = 0;
  routeLength = 0;
  thePath[0][0] = NULL;

  for(int i=0; i<5; i++)
    randSend_.reset_next_substream();
}
```

Gambar 4.13 Inisiasi variabel pada *class* GPSRAgent

4.4.4 Implementasi *Route Discovery* pada GPSR

Implementasi *route discovery* pada GPSR dilakukan dengan cara menambahkan RREQ dan RREP. Untuk menambahkan RREQ dan RREP pada GPSR dapat dilakukan pada *file* gpsr.cc pada fungsi **recv()**. Kode implementasi *route discovery* pada GPSR dapat dilihat pada lampiran A.6.

4.4.5 Implementasi Konsep *Overlay Network* pada RREP

Implementasi *overlay network* pada RREP dilakukan setelah melakukan implementasi *route discovery* pada GPSR. Saat sebuah node menerima paket RREP dan paket RREP tersebut bukan untuk node tersebut, maka node tersebut akan membandingkan posisi node tersebut dengan node sebelumnya. Apabila node tersebut dengan node sebelumnya memiliki perbedaan terhadap sumbu x dan sumbu y, maka posisi dari kedua node tersebut akan disimpan pada *header* RREP. Kode implementasi konsep *overlay network* dapat dilihat pada lampiran A.6.

4.4.6 Modifikasi *forwarding node* pada GPSR

Modifikasi *forwarding node* pada GPSR dilakukan dengan cara mengubah posisi *destination* pada *header* GPSR. Posisi *destination* sebelumnya merupakan posisi dari node *destination.* Setelah modifikasi, posisi *destination* pada *header* GPSR merupakan posisi yang harus dilalui paket data yang merupakan hasil dari RREP dan disimpan oleh node inisiator. Posisi-posisi yang disimpan node inisiator merupakan pengimplementasian dari konsep *overlay network* yang telah dibahas sebelumnya. Jika posisi pertama berhasil ditempuh paket data yang ditandai dengan paket data tersebut berhasil menemukan node disekitar posisi pertama, maka posisi *destination* akan di ubah menjadi posisi kedua, begitu seterusnya hingga paket data sampai node *destination*. Untuk lebih jelas mengenai modifikasi *forwarding node* pada GPSR dapat dilihat pada lampiran A.7.

4.5 Implementasi Simulasi pada NS-2

Implementasi simulasi pada NS-2 dilakukan pada *file-file* yang berekstensi *.tcl. Pada GPSR Ke Liu yang digunakan pada

Tugas Akhir ini terdapat contoh *file* tcl untuk melakukan simulasi percobaan GPSR. Penulis menggunakan *file* yang telah diberikan patch Ke Liu untuk dijadikan *file* TCL simulasi NS-2. *File* TCL dari Ke Liu kemudian dimodifikasi sesuai kebutuhan simulasi pada Tugas Akhir ini. *File-file* TCL yang digunakan pada Tugas Akhir ini dapat dilihat pada lampiran A.1 dan lampiran A.2

4.6 Implementasi Metrik Analisis

Penulis melakukan analisis menggunakan *file* hasil simulasi NS-2 yang berekstensi *.tr atau *tracefile*. Dari *tracefile* tersebut analisis dilakukan untuk mencari *packet delivery ratio* (PDR), *end to end delay*, dan *routing overhead*.

4.6.1 Implementasi *Packet Delivery Ratio* (PDR)

Implementasi *packet delivery ratio* dilakukan dengan cara membandingkan data yang dikirim dengan data yang diterima. Pada Tugas Akhir ini penulis menggunakan agen CBR untuk melakukan pengiriman paket data. Untuk mendapatkan nilai PDR dari suatu *tracefile* dapat dilakukan dengan menggunakan skrip AWK yang dapat dilihat pada lampiran A.8. Perintah untuk menjalankan skirp AWK adalah sebagai berikut:

```
awk -f PDR.awk trace.tr
```

4.6.2 Implementasi *End to End Delay*

Implementasi *end to end delay* dilakukan dengan cara menghitung rata-rata delay antara waktu paket data dikirimkan dengan waktu paket data diterima. Pada *tracefile* yang perlu diperhatikan adalah kolom pertama, kolom kedua, kolom keempat,

kolom keenam, dan kolom ketujuh. Untuk mendapatkan nilai *end to end delay* dari suatu *tracefile* dapat dilakukan dekan skrip AWK yang dapat dilihat pada lampiran A.9. Perintah untuk menjalankan skrip AWK adalah sebagai berikut:

```
awk -f e2e.awk trace.tr
```

4.6.3 Implementasi *Routing Overhead*

Implementasi *routing overhead* dilakukan dengan cara membagi antara jumlah paket *routing* dengan paket data. Paket data pada *tracefile* dapat dilihat pada baris yang memiliki tulisan cbr dan pada kolom pertama terdapat huruf r yang artinya *received*. Paket *routing* pada tracefile dapat dilihat pada baris yang memiliki tulisan AGT, gpsr, dan pada kolom pertamanya terdapat huruf s maupun r. Untuk mendapatkan nilai *routing overhead* dari suatu *tracefile* dapat dilakukan dengan menggunakan skrip AWK yang dapat dilihat pada lampiran A.10. Perintah untuk menjalankan skrip AWK adalah sebagai berikut:

```
awk -f r_overhead.awk trace.tr
```

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan membahas uji coba dan evaluasi tentang skenario-skenario yang telah disimulasikan di NS-2.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi seperti yang dijabarkan pada Tabel 5.1.

Tabel 5.1 Spesifikasi perangkat uji coba

Komponen	Spesifikasi
CPU	Intel core i7-6700hq @2.40GHz
Sistem Operasi	Ubuntu 14.04 LTS 64-bit
Memori	4GB Ram
Penyimpanan	40GB

Uji coba dilakukan dengan menjalankan skenario-skenario yang telah dibuat dan disimulasikan pada NS-2. Hasil dari simulasi berupa *tracefile* yang akan digunakan untuk menganalisis nilai *packet delivery ratio* (PDR), *end to end delay*, dan *routing overhead* dengan bantuan skrip AWK.

5.2 Skenario Uji Coba

Skenario uji coba memiliki parameter-parameter seperti yang dijabarkan pada Tabel 5.2.

Tabel 5.2 Parameter skenario pengujian

No.	Parameter	Spesifikasi
1	*Network Simulator*	NS-2.35
2	*Routing Protocol*	GPSR, GODV
3	Waktu Simulasi	200 detik

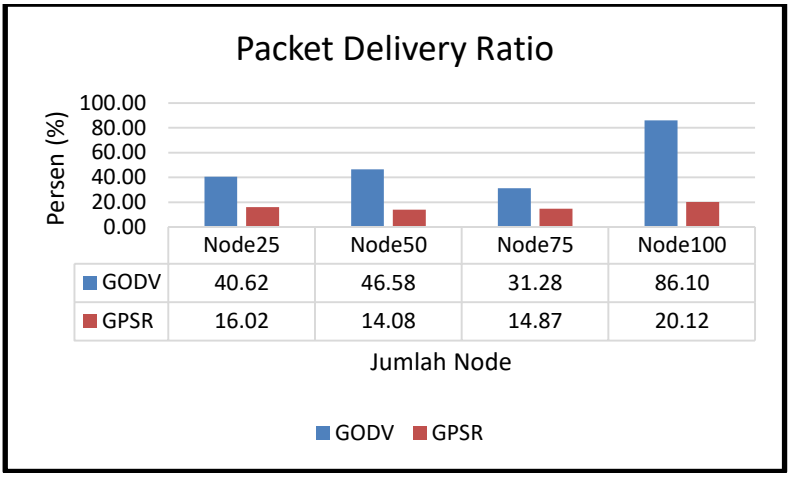
No.	Parameter	Spesifikasi
4	Area Simulasi	*Grid*: 800 m x 800 m *Real*: 800m x 800 m
5	Kecepatan	10 m/s, 15 m/s, 20m/s
6	Banyak Node	25, 50, 75, 125
7	Radius Transmisi	250m
8	*Source* / *Destination*	statis
9	Agen Pengirim	CBR
10	Ukuran Paket	32 byte
11	Protokol MAC	802.11
12	Propagasi Sinyal	*Two-ray ground*
13	Tipe Kanal	*Wireless channel*

5.3 Hasil Uji Coba Skenario *Grid*

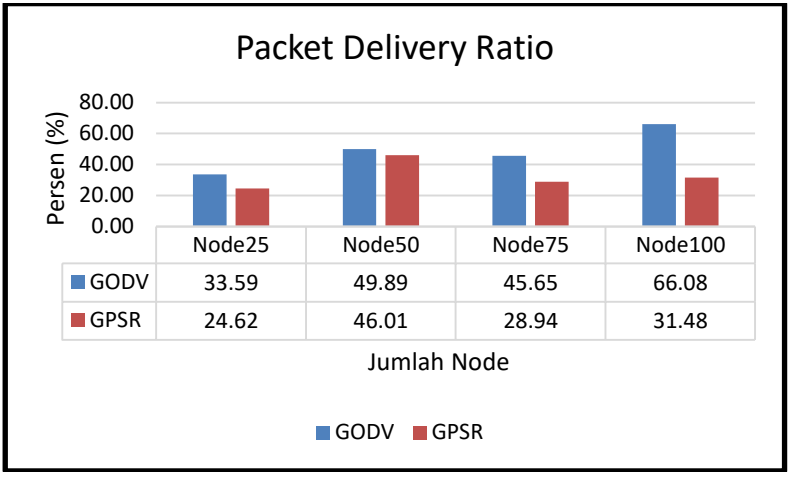
Pengujian skenario *grid* dilakukan dengan parameter-parameter yang dapat dilihat pada Tabel 5.2. Luas area simulasi adalah 800x800 meter dengan variasi node sebanyak 25, 50, 75, dan 100 pada variasi kecepatan 10 m/s, 15 m/s, dan 20 m/s. Untuk setiap banyak node pada kecepatan yang sama dilakukan pada 5 simulasi yang berbeda yang kemudian dihitung rata-rata nilainya.

5.3.1 Hasil *Packet Delivery Ratio* pada Skenario *Grid*

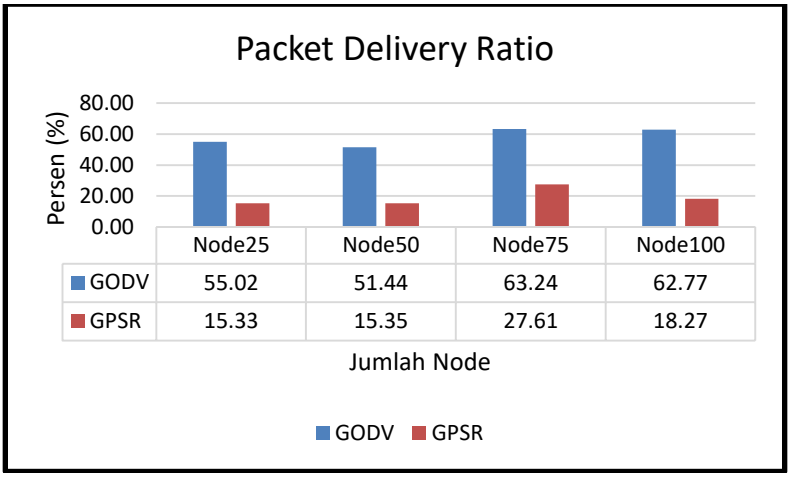
Hasil *packet delivery ratio* berdasarkan Gambar 5.1, Gambar 5.2, dan Gambar 5.3 GODV memiliki rata-rata PDR lebih tinggi daripada GPSR pada variasi node yang diterapkan yaitu 25, 50, 75, dan 100 pada variasi kecepatan yang berbeda, yaitu 10 m/s, 15m/s, dan 20 m/s. GODV memiliki PDR dengan rata-rata di atas 50% dan GPSR dengan rata-rata di bawah 50%.



Gambar 5.1 Grafik PDR pada kecepatan 10 m/s



Gambar 5.2 Grafik PDR pada kecepatan 15 m/s



Gambar 5.3 Grafik PDR pada kecepatan 20 m/s

Pada grafik dengan kecepatan 15 m/s pada node sebanyak 50 ada dua simulasi di mana protokol GODV kalah dengan GPSR dikarenakan posisi node yang memang *random* dan kemungkinan GODV mendapatkan rute pengiriman paket yang jarang ditemukan node.

Tabel 5.3 Simulasi pada node 50 dengan kecepatan 15 m/s

Simulasi ke-	GODV	GPSR
1	48.40 %	89.95 %
2	35.64 %	4.49 %
3	94.02 %	43.01 %
4	40,88 %	23,12 %
5	30,48 %	69,49 %

Tabel 5.3 menunjukkan bahwa tidak selalu protokol GODV yang unggul pada simulasi yang dilakukan. Untuk mengetahui kenapa PDR pada GODV lebih kecil daripada GPSR perlu dilakukan analisis pada *tracefile* milik GODV dan *tracefle* milik GPSR. Pada *tracefile* terlihat bahwa node *source* tidak

mendapatkan *neighbor* yang lebih dekat dengan *destination* seperti yang ditunjukkan Gambar 5.4.

```
- s 51.488426133 _51_ AGT  --- 2811 cbr 32 [0 0 0 0] ------
  - [51:0 50:0 32 0] [45] 0 0
- r 51.488426133 _51_ RTR  --- 2811 cbr 32 [0 0 0 0] -------
  [51:0 50:0 32 0] [45] 0 0
- s 51.488426133 _51_ RTR  --- 2811 cbr 83 [0 0 0 0] -------
  [51:0 50:0 31 0] [45] 0 0
```

Gambar 5.4 Node tidak menemukan *nexthop*

```
- s 60.630136257 _51_ AGT  --- 3291 cbr 32 [0 0 0 0] ------
  - [51:0 50:0 32 0] [53] 0 0
- r 60.630136257 _51_ RTR  --- 3291 cbr 32 [0 0 0 0] -------
  [51:0 50:0 32 0] [53] 0 0
- s 60.630136257 _51_ RTR  --- 3291 cbr 83 [0 0 0 0] ------
  - [51:0 50:0 31 12] [53] 0 0
- r 60.634956468 _12_ RTR  --- 3291 cbr 83 [13a c 33 800]
  ------- [51:0 50:0 31 12] [53] 1 0
- f 60.634956468 _12_ RTR  --- 3291 cbr 83 [13a c 33 800]
  ------- [51:0 50:0 30 50] [53] 1 0
- r 60.640200264 _50_ AGT  --- 3291 cbr 83 [13a 32 c 800]
  ------- [51:0 50:0 30 50] [53] 2 0
```

Gambar 5.5 Node menemukan *nexthop*

Pada Gambar 5.4, node 51 menginisiasi pengiriman paket data pada detik 51 yang ternyata tidak ditemukan *neighbor*. Gambar 5.5 menunjukkan bahwa pada detik ke-60 node 51 baru menemukan *nexthop* untuk *forwarding data* menuju node *destination*. Gambar 5.4 dan Gambar 5.5 merupakan salah satu

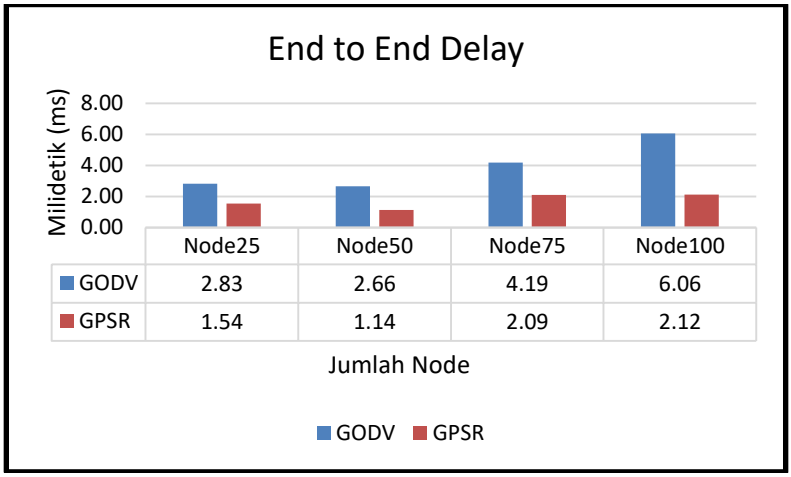
contoh yang menyebabkan PDR GODV pada simulasi pertama dan ke lima lebih kecil daripada GPSR.

Rata-rata PDR berdasarkan Gambar 5.1, Gambar 5.2, dan Gambar 5.3 untuk protokol GODV adalah 52.7 %, sedangkan untuk protokol GPSR adalah 22.7 %.

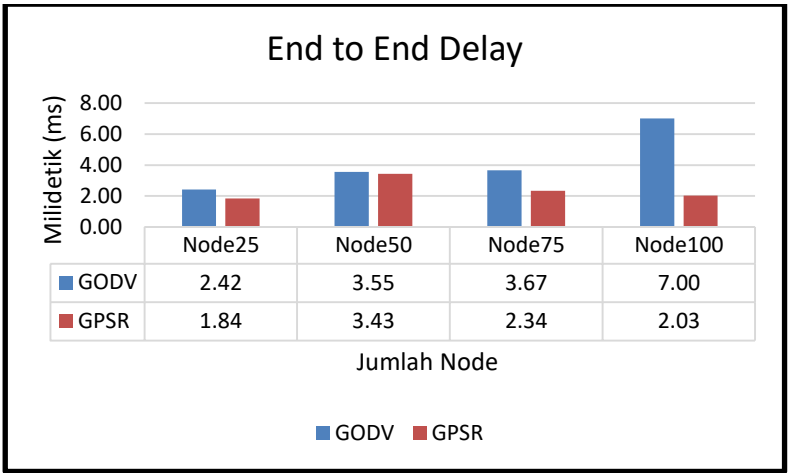
5.3.2 Hasil *End to End Delay* pada Skenario *Grid*

Gambar 5.6, Gambar 5.7, dan Gambar 5.8 menjelaskan grafik *end to end delay* dengan variasi node 25, 50, 75, dan 100 pada variasi kecepatan 10 m/s, 15 m/s, dan 20 m/s.

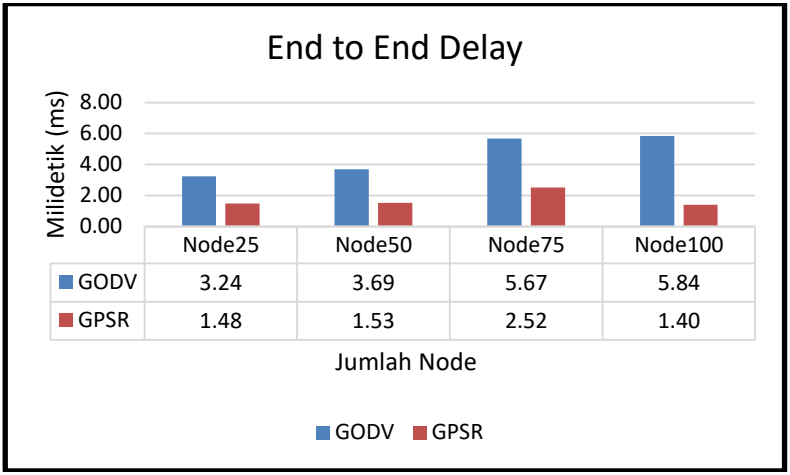
Gambar 5.6 merupakan grafik hasil *end to end delay* dengan kecepatan 10 m/s. Pada protokol GPSR rata-rata *end to end delay* untuk setiap variasi node ada di bawah 2 ms, dan pada protokol GODV rata-rata *end to end delay* untuk setiap variasi node ada di atas 2 ms. *End to end delay* dari node 25 ke node 50 mengalami penurunan pada kedua protokol. Mulai dari node 50 hingga node 100 baik GODV maupun GPSR mengalami kenaikan *end to end delay*.



Gambar 5.6 Grafik *end to end delay* pada kecepatan 10 m/s



Gambar 5.7 Grafik *end to end delay* pada kecepatan 15 m/s



Gambar 5.8 Grafik *end to end delay* pada kecepatan 20 m/s

Gambar 5.7 merupakan grafik hasil *end to end delay* dengan kecepatan 15 m/s. *End to end delay* pada protokol GODV terlihat semakin tinggi apabila jumlah node semakin tinggi, meskipun terjadi penuruan pada node 75. Pada protokol GPSR *end*

to end delay hanya mengalami kenaikan pada node 50, selanjutnya *end to end delay* terlihat menurun hingga node 100. *End to end delay* pada GPSR terlihat bertolak belakang dengan GODV.

Gambar 5.8 merupakan grafik hasil *end to end delay* dengan kecepatan 20 m/s. *End to end delay* pada protokol GODV mengalami kenaikan seiring bertambahnya jumlah node yang ada. Pada GPSR mengalami kenaikan yang sama dengan protokol GODV, tetapi pada node 100 *end to end delay* protokol GPSR mengalami penurunan dibandingkan dengan node 75.

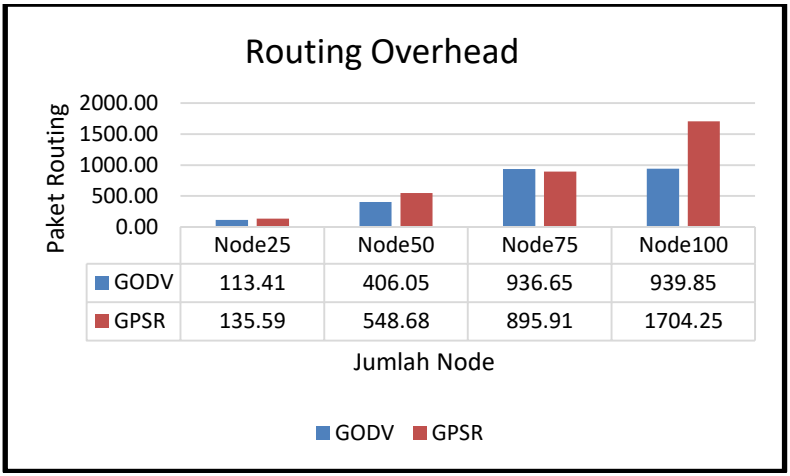
Dari ketiga hasil *end to end delay* pada kecepatan 10 m/s, 15 m/s, dan 20 m/s, protokol GODV memang lebih lama dari protokol GPSR. Hal ini dikarenakan rute perjalanan yang ada pada GODV memang belum tentu yang terdekat, melainkan rute tercepat yang didapat berdasarkan *route discovery* yang dilakukan. GPSR mungkin secara *end to end delay* lebih baik dari protokol GODV, karena GPSR benar-benar mencari *neighbor* yang terdekat dengan *destination* secara *greedy*.

Rata-rata *end to end delay* berdasarkan Gambar 5.6, Gambar 5.7, dan Gambar 5.8 untuk protokol GODV adalah 4.24 ms, dan untuk protokol GPSR adalah 1.96 ms.

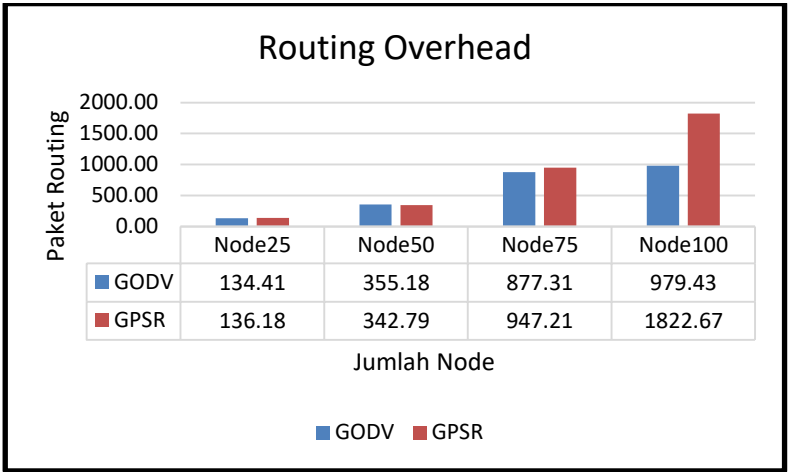
5.3.3 Hasil *Routing Overhead* pada Skenario *Grid*

Gambar 5.9, Gambar 5.10, dan Gambar 5.11 menjelaskan grafik *routing overhead* terhadap variasi node 25, 50, 75, dan 100 pada variasi kecepatan 10 m/s, 15 m/s, dan 20 m/s.

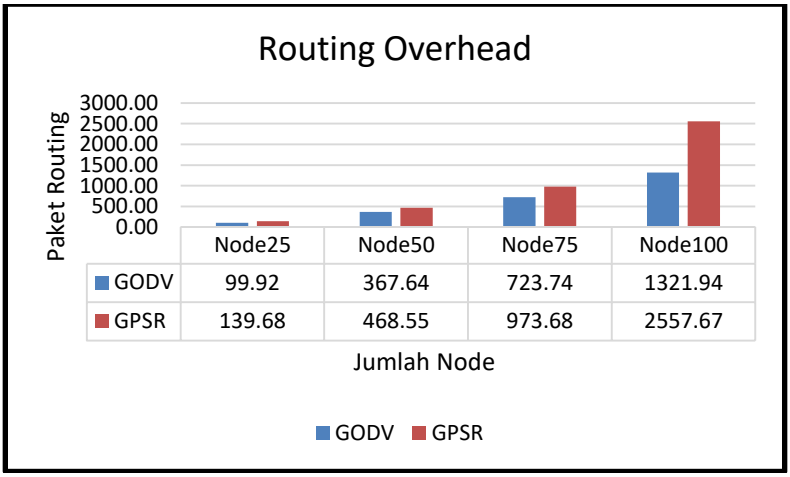
Gambar 5.9 merupakan grafik *routing overhead* dengan kecepatan 10 m/s. Dari Gambar 5.9 terlihat bahwa pada protokol GODV *routing overhead* mengalami kenaikan seiring bertambahnya jumlah node yang ada. Hal yang sama pun terjadi pada protokol GPSR. Secara keseluruhan pada grafik tersebut menunjukkan bahwa *routing overhead* pada protokol GODV lebih kecil daripada protokol GPSR, tetapi hanya pada node 75 protokol GODV sedikit lebih tinggi dibandingkan dengan protokol GPSR.



Gambar 5.9 Grafik *routing overhead* pada kecepatan 10 m/s



Gambar 5.10 Grafik *routing overhead* pada kecepatan 15 m/s



Gambar 5.11 Grafik *end to end delay* pada kecepatan 20 m/s

Gambar 5.10 merupakan grafik *routing overhead* pada kecepatan 15 m/s. Grafik menunjukkan bahwa baik itu prokol GODV maupun GPSR *routing overhead* tiap protokol mengalami kenaikan seiring bertambahnya jumlah node yang ada. Pada grafik ditunjukkan pula bahwa pada node 25, 75, dan 100 protokol GODV memiliki nilai *routing overhead* yang lebih kecil dibandingkan dengan protokol GPSR. Hanya pada node 50 protokol GPSR memiliki nilai *routing overhead* yang lebih kecil dibandingkan dengan protokol GODV.

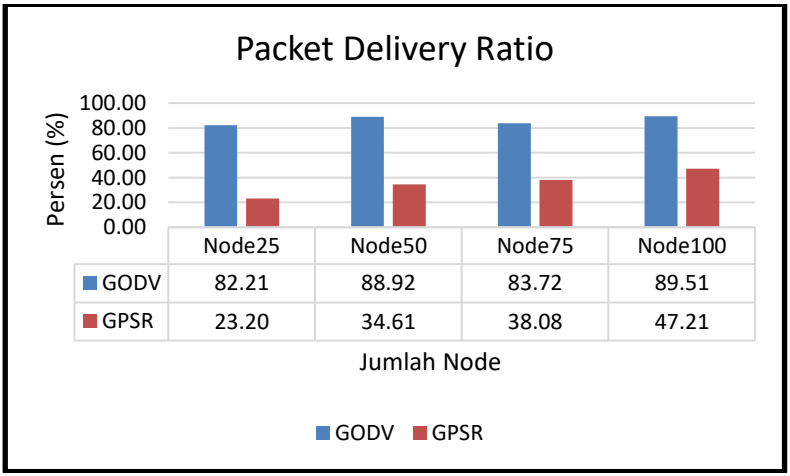
Gambar 5.11 merupakan grafik *routing overhead* pada kecepatan 20 m/s. Grafik menunjukkan bahwa pada kedua protokol baik itu GODV maupun GPSR, nilai *routing overhead* semakin meningkat seiring bertambahnya jumlah node yang ada. Pada setiap jumlah node yang di uji, protokol GODV selalu memiliki nilai *routing overhead* yang lebih kecil dibandingkan dengan protokol GPSR.

Rata-rata *routing overhead* berdasarkan Gambar 5.9, Gambar 5.10, dan Gambar 5.11, untuk protokol GODV adalah 604.63 paket dan untuk protokol GPSR adalah 889.41 paket.

5.4 Hasil Uji Coba Skenario *Real*

Pengujian skenario *real* dilakukan untuk melihat performa protokol GODV jika diimplementasikan menggunakan peta dunia nyata yang mana bentuk jalannya tidak seperti pada skenario *grid*. Pengujian dilakukan pada luas area 800x800 meter dengan variasi node sebanyak 25, 50, 75, dan 100 pada kecepatan standar sesuai dengan tipe jalan pada wilayah peta yang digunakan.

5.4.1 Hasil *Packet Delivery Ratio* pada Skenario *Real*

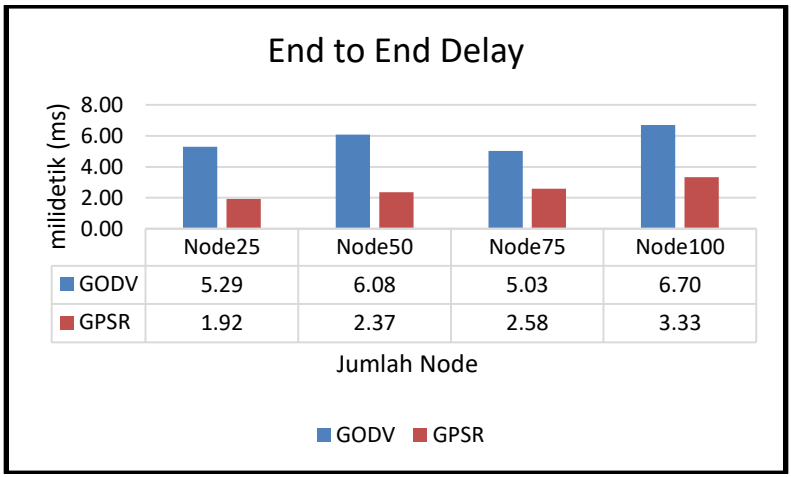


Gambar 5.12 Grafik *packet delivery ratio* pada skenario *real*

Gambar 5.12 merupakan grafik yang memaparkan nilai *packet delivery ratio* pada pengujian skenario *real*. Dari Grafik di atas dapat dilihat bahwa pada protokol GPSR nilai *packet delivery ratio* meningkat seakan bertambahnya jumlah node yang ada. Hal yang berbeda terlihat pada protokol GODV yang mengalami naik turun. GODV mengalami peningkatan *packet delivery ratio* dari

82.2 % pada node 25 menjadi 88.92 % pada node 50, kemudian mengalami penurunan pada node 75 dengan nilai 83.7 % dan akhirnya pada node 100 naik kembali dengan nilai 89.5 %.

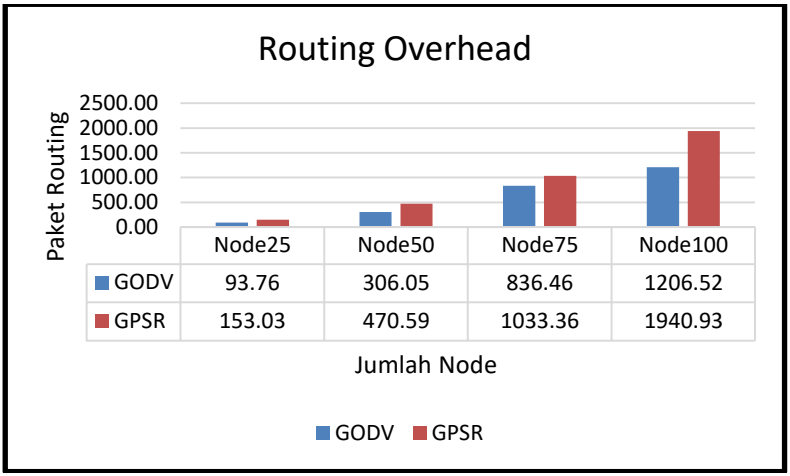
5.4.2 Hasil *End to End Delay* pada Skenario *Real*



Gambar 5.13 Grafik *end to end delay* pada skenario *real*

Gambar 5.13 merupakan grafik yang memaparkan nilai *end to end delay* pada pengujian skenario *real*. Nilai *end to end delay* protokol GPSR terlihat lebih baik dibandingkan dengan protokol GODV. Nilai *end to end delay* yang lebih baik belum menentukan bahwa protokol GPSR lebih baik dibandingkan dengan protokol GODV. Nilai *end to end delay* GODV lebih tinggi karena GODV mendapatkan rute perjalanan paket data yang lebih lama dibandingkan dengan protokol GPSR yang pasti mencari yang paling dekat dengan node *destination*.

5.4.3 Hasil *Routing Overhead* pada Skenario *Real*



Gambar 5.14 Grafik *routing overhead* pada skenario *real*

Gambar 5.14 menunjukkan bahwa terjadi peningkatan nilai *routing overhead* seiring bertambahnya node yang ada, baik pada protokol GODV maupun protokol GPSR. Nilai *routing overhead* pada protokol GODV selalu lebih kecil pada setiap variasi node yang di uji cobakan. Hal ini menunjukkan bahwa protokol GODV memiliki nilai *routing overhead* yang lebih baik dibandingkan dengan protokol GPSR.

[Halaman ini sengaja dikosongkan]

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan dibahas mengenai kesimpulan yang dapat diambil dari pengujian yang telah dilakukan dan saran yang ditujukan untuk pengembangan lebih lanjut terhadap Tugas Akhir ini.

6.1 Kesimpulan

Setelah melakukan pengujian dan evaluasi, didapatkan kesimpulan sebagai berikut:

1. *Packet Delivery Ratio* (PDR) menunjukkan bahwa pada skenario *grid* maupun skenario *real*, protokol GODV lebih baik dibandingkan dengan protokol GPSR. Pada skenario *grid*, rata-rata nilai PDR untuk protokol GODV adalah 52.69 % sedangkan rata-rata nilai PDR untuk protokol GPSR adalah 22.73 %. Pada skenario *real*, rata-rata nilai PDR untuk protokol GODV adalah 86.09 % sedangkan rata-rata nilai PDR untuk protokol GPSR adalah 35.78 %.
2. *End to end delay* pada protokol GODV lebih tinggi dibandingkan dengan *end to end delay* protokol GPSR, hal ini dikarenakan rute perjalanan yang dihasilkan dari *route discovery* tidak selalu rute yang tercepat. Pada skenario *grid*, rata-rata nilai *end to end delay* untuk protokol GODV adalah 4.24 ms dan rata-rata nilai *end to end delay* untuk protokol GPSR adalah 1.96 ms. Pada skenario *real*, rata-rata nilai *end to end delay* pada protokol GODV adalah 5.77 ms dan rata-rata nilai *end to end delay* pada protokol GPSR adalah 2.55 ms.
3. *Routing Overhead* pada protokol GODV lebih baik daripada *routing overhead* protokol GPSR karena kesuksesan pengiriman paket data lebih baik pada protokol GODV. Pada skenario *grid*, rata-rata nilai *routing overhead* untuk protokol GODV adalah 604.63 paket, sedangkan untuk protokol GPSR adalah 889.41 paket. Pada skenario *real*, rata-rata nilai *routing*

overhead untuk protokol GODV adalah 610.70 paket, sedangkan untuk protokol GPSR adalah 899.48 paket.

4. Implementasi *route discovery* dan konsep *overlay network* dapat meningkatkan rasio kesuksesan pengiriman paket data pada VANETs yang menyebabkan turunnya nilai *routing overhead*, namun nilai *end to end delay* akan meningkat.
5. Rute yang didapatkan dari proses *route discovery* belum tentu rute yang terbaik karena akan ada kondisi di mana rute yang didapatkan tidak terdapat node untuk melakukan pengiriman paket data.

6.2 Saran

Adapun saran-saran yang diberikan untuk pengembangan sistem ini kedepannya adalah sebagai berikut:

1. Pada saat melakukan perbandingan antara dua posisi node pada paket RREP dapat ditambahkan perhitungan untuk mengetahui lebih spesifik apakah dua buah node berada pada bidang jalan yang berbeda.
2. Pada node inisiator dapat di implementasikan rute perjalanan menuju lebih dari satu *destination*.
3. *Maintenance* rute perjalanan dapat ditambahkan untuk melakukan *route discovery* ulang apabila node *source* dan *destination* dinamis.

DAFTAR PUSTAKA

[1] S. P. Aruna Sharma, "Node Selection Algorithm for Routing Protocols in VANET," *International Journal of Advanced Science and Technology,* vol. 96, pp. 43-54, 2016.

[2] N. T. Silmi, Modifikasi Protokol GPSR-MV dalam Pemilihan Forwarding Node pada VANET, Buku Tugas Akhir Teknik Informatika, FTIF-ITS, 2016.

[3] P. A. R. P. S. P. Nimpal Patel, "A Survey Paper on Dynamic Source Routing Protocol (DSR) in Ad-Hoc Network," *International Journal for Scientific Research & Development - IJSRD,* vol. 2, no. 10, pp. 43-45, 2014.

[4] J. G.-J. a. A. Gazo-Cervero, "Overview and Challenges of Overlay Network: A Survey," *International Journal of Computer Science & Engineering Survey (IJCSES)* , vol. 2, no. 1, pp. 19-37, 2011.

[5] "SUMO - Simulation of Urban MObility," [Online]. Available: http://www.sumo.dlr.de/wiki/Simulation_of_Urban_MObility_-_Wiki. [Accessed 10 May 2017].

[6] "OpenStreetMap," [Online]. Available: http://www.openstreetmap.org. [Accessed 31 May 2017].

[7] "JOSM," [Online]. Available: http://wiki.openstreetmap.org/wiki/JOSM. [Diakses 31 May 2017].

[8] "The GNU Awk User's Guide," [Online]. Available: https://www.gnu.org/software/gawk/manual/gawk.html. [Diakses 2 June 2017].

[9] "NS-2 Simulator," [Online]. Available: http://www.isi.edu/nsnam/ns/. [Accessed 5 April 2017].

[10] R. Bastian, Implementasi Source Route pada Protokol GPSR dengan Bantuan Intersection Node untuk Meningkatkan Reliabilitas Pengiriman Data di VANET, Buku Tugas Akhir Teknik Informatika, FTIF-ITS, 2017.

LAMPIRAN

A.1. Skenario GPSR.tcl

```
set opt(chan)        Channel/WirelessChannel
set opt(prop)        Propagation/TwoRayGround
set opt(netif)       Phy/WirelessPhy
set-
 opt(mac)            Mac/802_11
set opt(ifq)         Queue/DropTail/PriQueue    ;#
for dsdv
set opt(ll)          LL
set opt(ant)         Antenna/OmniAntenna

set opt(x)           802    ;# X
set opt(y)           802    ;# Y
set opt(cp)          "./godvCBRtest.tcl"
set opt(sc)          "./GODVnodePos.tcl"
set opt(ifqlen)      512    ;# max packet in ifq
set opt(nn)          52            ;# number of nodes
set opt(seed)        0.0
set opt(stop)        200.0         ;# simulation time
set opt(tr)          trace.tr      ;# trace file
set opt(nam)         nam.out.tr
set opt(rp)          gpsr          ;
set opt(lm)          "off"         ;# log movement

Phy/WirelessPhy set Pt_ 0.2818    ; #250meter

# Agent/GPSR setting
Agent/GPSR set planar_type_  1   ;#1=GG planarize,
0=RNG planarize
Agent/GPSR set hello_period_   1.0 ;#Hello message
period

proc usage { argv0 }  {
       puts "Usage: $argv0"
       puts "\tmandatory arguments:"
       puts "\t\t\[-x MAXX\] \[-y MAXY\]"
       puts "\toptional arguments:"
       puts "\t\t\[-cp conn pattern\] \[-sc
scenario\] \[-nn nodes\]"
```

```
       puts "\t\t\[-seed seed\] \[-stop sec\] \[-tr
tracefile\]\n"
}


proc getopt {argc argv} {
       global opt
       lappend optlist cp nn seed sc stop tr x y

       for {set i 0} {$i < $argc} {incr i} {
              set arg [lindex $argv $i]
              if {[string range $arg 0 0] != "-"}
continue

              set name [string range $arg 1 end]
              set opt($name) [lindex $argv [expr
$i+1]]
       }
}

proc log-movement {} {
    global logtimer ns_ ns

    set ns $ns_
    source ../tcl/mobility/timer.tcl
    Class LogTimer -superclass Timer
    LogTimer instproc timeout {} {
       global opt node_;
       for {set i 0} {$i < $opt(nn)} {incr i} {
           $node_($i) log-movement
       }
       $self sched 0.1
    }

    set logtimer [new LogTimer]
    $logtimer sched 0.1
}

source ../tcl/lib/ns-bsnode.tcl
source ../tcl/mobility/com.tcl

# do the get opt again incase the routing protocol
file added some more
# options to look for
```

```
getopt $argc $argv

if { $opt(x) == 0 || $opt(y) == 0 } {
       usage $argv0
       exit 1
}

if {$opt(seed) > 0} {
       puts "Seeding Random number generator with
$opt(seed)\n"
       ns-random $opt(seed)
}

#
# Initialize Global Variables
#
set ns_              [new Simulator]
set chan      [new $opt(chan)]
set prop      [new $opt(prop)]
set topo      [new Topography]

set tracefd   [open $opt(tr) w]
$ns_ trace-all  $tracefd

set namfile [open $opt(nam) w]
$ns_ namtrace-all $namfile

$topo load_flatgrid $opt(x) $opt(y)

$prop topography $topo

#
# Create God
#
set god_ [create-god $opt(nn)]

$ns_ node-config -adhocRouting gpsr \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif) \
```

```
                    -channelType $opt(chan) \
                    -topoInstance $topo \
                    -agentTrace ON \
                    -routerTrace ON \
                    -macTrace OFF \
                    -movementTrace OFF

source ./gpsr.tcl

for {set i 0} {$i < $opt(nn) } {incr i} {
    gpsr-create-mobile-node $i
}


#
# Source the Connection and Movement scripts
#
if { $opt(cp) == "" } {
       puts "*** NOTE: no connection pattern
specified."
        set opt(cp) "none"
} else {
       puts "Loading connection pattern..."
       source $opt(cp)
}

#
# Tell all the nodes when the simulation ends
#
for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).000000001 "$node_($i)
reset";
}
$ns_ at $opt(stop).00000001 "puts \"NS
EXITING...\" ; $ns_ halt"


if { $opt(sc) == "" } {
       puts "*** NOTE: no scenario file specified."
         set opt(sc) "none"
} else {
       puts "Loading scenario file..."
       source $opt(sc)
       puts "Load complete..."
```

```
}

puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y
$opt(y) rp $opt(rp)"
puts $tracefd "M 0.0 sc $opt(sc) cp $opt(cp) seed
$opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"

puts "Starting Simulation..."

proc finish {} {
    global ns_ tracefd namfile
    $ns_ flush-trace
    close $tracefd
    close $namfile
    exit 0
}

$ns_ at $opt(stop) "finish"

$ns_ run
```

A.2. Skenario Pengiriman Paket Data

```
# GPSR routing agent settings
for {set i 0} {$i < $opt(nn)} {incr i} {
    $ns_ at 0.00002 "$ragent_($i) turnon"
    $ns_ at 3.0 "$ragent_($i) neighborlist"
#    $ns_ at 149.0 "$ragent_($i) turnoff"
#      $ns_ at 80 "$ragent_($i) turnon"
}

$ns_ at 3.0 "$ragent_(50) startSink 10.0"

set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(50) $null_(0)

set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(51) $udp_(0)

set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 32
$cbr_(0) set interval_ 1.0
$cbr_(0) set random_ 1
#    $cbr_(0) set maxpkts_ 100
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 10.0 "$cbr_(0) start"
$ns_ at 195.0 "$cbr_(0) stop"
```

A.3. Kode *Routing Table*

```
#include <gpsr/godv_rtable.h>
/*
  The Routing Table
*/
godv_rtable::godv_rtable()
{
       rt = new godv_rt_entry[128];
       len = 0;
}

int godv_rtable::rt_lookup(nsaddr_t id)
{
       for(int i=0; i<len; i++){
              if(rt[i].rt_id == id){
                     return i;
              }
       }
       return len;
}

int godv_rtable::rt_getrid(nsaddr_t id){
       int param = rt_lookup(id);
       if(param >= len){
              return 0;
       }
       return rt[param].rt_reqno;
}

void godv_rtable::rt_add(nsaddr_t id, int reqno)
{
       int param = rt_lookup(id);

       if(param < len){
              rt[param].rt_reqno = reqno;
       }

       rt[len].rt_id = id;
       rt[len].rt_reqno = reqno;
       len++;
}
```

A.4. Kode *Check Request Packet*

```
bool
GPSRAgent::checkRequest(Packet *p)
{
  struct hdr_ip *iph = HDR_IP(p);
  struct hdr_godv_data *godv = HDR_GODV_DATA(p);

  if(rt->rt_getrid(iph->saddr()) >= godv->rq_id){
    return true;
  }

  if(godv->rq_ttl == 0){ //TTL Expire
    return true;
  }

  if(iph->saddr() == my_id_){//I'm the source
    return true;
  }

  for(int i=0; i < godv->length; i++){//Already on
route path
    if(godv->nodePath[i] == my_id_)
      return true;
  }
  return false;//Request OK
}
```

A.5. Kode Pengiriman RREP

```
void
GPSRAgent::sendReply(Packet *p)
{
  struct hdr_ip *iph = HDR_IP(p);
  struct hdr_godv_data *godv = HDR_GODV_DATA(p);

  //Create new RREP packet
  Packet *rp = allocpkt();
  struct hdr_cmn *rcmh = HDR_CMN(rp);
  struct hdr_ip *riph = HDR_IP(rp);
  struct hdr_godv_data *rgodv = HDR_GODV_DATA(rp);

  riph->saddr() = my_id_;
  riph->sport() = RT_PORT;
  riph->daddr() = iph->saddr();
  riph->dport() = RT_PORT;
  riph->ttl()   = 255;

  //GODV header
  rgodv->pValid = 1;
  rgodv->repValid = 1;
  rgodv->rp_id = godv->rq_id;
  rgodv->type_ = GPSRTYPE_ROUTEDISC;
  rgodv->length = godv->length;
  rgodv->pathLen = 0;
  int param = 0;
  for(int i= rgodv->length; i>=0; i--){
    rgodv->nodePath[param]=godv->nodePath[i];
    param++;
  }

  //insert Loc of Destination
  MobileNode      *tempNode      =      (MobileNode
*)(Node::get_node_by_address(my_id_));
  rgodv->path[rgodv->pathLen][0] = tempNode->X();
  rgodv->path[rgodv->pathLen][1] = tempNode->Y();
  rgodv->overlayNode[rgodv->pathLen] = my_id_;
  rgodv->pathLen++;

  //rcmh->next_hop_   =   nblist_->gf_nexthop(rgodv-
>path[rgodv->index][0],rgodv->path[rgodv-
>index][1]);
```

```
  rgodv->index = 1;
  rcmh->next_hop_   = rgodv->nodePath[rgodv->index];
  rcmh->last_hop_   = my_id_;
  rcmh->addr_type_  = NS_AF_INET;
  rcmh->ptype()     = PT_GPSR;
  rcmh->size()      = IP_HDR_LEN+rgodv->size();
  rcmh->direction() = hdr_cmn::DOWN;
  Scheduler::instance().schedule(this, rp, 0.010);
}
```

A.6. Kode Implementasi *Route Discovery*

```
void
GPSRAgent::recv(Packet *p, Handler *h){
  struct hdr_cmn *cmh = HDR_CMN(p);
  struct hdr_ip *iph = HDR_IP(p);
  struct hdr_godv_data *godv = HDR_GODV_DATA(p);

  if(godv->pValid == 0){ //godv header valid
    if(iph->saddr() == my_id_){ //I'm the Source
      if(valid_route == 1){ //I have valid route
        if(src != iph->saddr() && dst != iph-
>daddr()){ //Destination route is not legit
          valid_route = 0; //make Route Discovery
to Destination
        }
      }
      if(valid_route == 0){ //Route Discovery
        godv->pValid = 1; //godv header valid
        //header RREQ
        godv->reqValid = 1;
        godv->rq_id = rq_id++;
        godv->rq_ttl = 20;
        godv->length = 0;
        godv->index = 0;
        godv->nodePath[godv->length] = my_id_;
        godv->length++;

        godv->type_ = GPSRTYPE_ROUTEDISC; //type

        cmh->next_hop_       = IP_BROADCAST;
        cmh->last_hop_       = my_id_;
        cmh->addr_type_      = NS_AF_INET;
        cmh->ptype()         = PT_GPSR;
        cmh->size()          = IP_HDR_LEN+godv-
>size();
        cmh->direction()     = hdr_cmn::DOWN;
        cmh->num_forwards()  = 0;

        iph->saddr()         = my_id_;
        iph->sport()         = RT_PORT;
        iph->daddr()         = iph->daddr();
        iph->dport()         = RT_PORT;
      }
```

```
      //forward data
      else{
        if(iph->saddr() == my_id_){//a packet
generated by myself
          if(cmh->num_forwards() == 0){
            struct hdr_gpsr_data *gdh =
HDR_GPSR_DATA(p);
            cmh->size() += IP_HDR_LEN + gdh->size()
+ godv->size();

            //godv data
            godv->pValid = 0;
            godv->length = routeLength;
            godv->index = 0;
            for(int i=0; i<routeLength;i++){
              for(int j=0; j<2; j++){
                godv->path[i][j] = thePath[i][j];
              }
            }

            //gpsr data
            gdh->type_ = GPSRTYPE_DATA;
            gdh->mode_ = GPSR_MODE_GF;
            gdh->sx_ = (float)my_x_;
            gdh->sy_ = (float)my_y_;
            gdh->dx_ = (float)godv->path[godv-
>index][0];
            gdh->dy_ = (float)godv->path[godv-
>index][1];
            gdh->ts_ = (float)GPSR_CURRENT;
          }
          else if(cmh->num_forwards() >
0){ //routing loop
            if(cmh->ptype() != PT_GPSR)
             drop(p, DROP_RTR_ROUTE_LOOP);
            else Packet::free(p);
            return;
          }
        }
      }
    }
  }
  else if(godv->pValid == 1) //godv header valid
  {
```

```
    if(iph->daddr() == my_id_) { //I'm the
Destination
      if(godv->reqValid){ //RREQ
        godv->nodePath[godv->length] = my_id_;
        sendReply(p);
        goto done;
      }
      else if(godv->repValid){ //RREP
        if(valid_route == 1){
          goto done;
        }

        MobileNode *prevNode = (MobileNode
*)(Node::get_node_by_address(cmh->last_hop_));
        MobileNode *thisNode = (MobileNode
*)(Node::get_node_by_address(my_id_));
        if(prevNode->X() != thisNode->X() &&
prevNode->Y() != thisNode->Y()){//check
intersection
          if(godv->overlayNode[godv->pathLen-1] !=
cmh->last_hop_){ //check Overlay Node list
            godv->path[godv->pathLen][0] =
prevNode->X();
            godv->path[godv->pathLen][1] =
prevNode->Y();
            godv->overlayNode[godv->pathLen] = cmh-
>last_hop_;
          }
        }
        else{
          godv->pathLen--;
        }

        int temp = 0;
        if(godv->pathLen == 0){
          thePath[godv->pathLen][0] = godv-
>path[godv->pathLen][0];
          thePath[godv->pathLen][1] = godv-
>path[godv->pathLen][1];
          temp++;
        }
        else{
          for(int i=godv->pathLen-1; i>=0; i--){
            for(int j=0; j<2; j++){
```

```
              thePath[temp][j] = godv->path[i][j];
            }
            temp++;
          }
        }

        routeLength = temp;
        src = godv->nodePath[godv->length];
        dst = godv->nodePath[0];
        valid_route =1;
        goto done;
      }
    }
    else{ //I'm not the Destination
      if(godv->reqValid){ //RREQ
        cmh->num_forwards()++;
        godv->rq_ttl--;
        if(checkRequest(p)){
          Packet::free(p);
          goto done;
        }
        MobileNode *lastNode = (MobileNode
*)(Node::get_node_by_address(cmh->last_hop_));
        nblist_->newNB(cmh->last_hop_, lastNode-
>X(), lastNode->Y());
        //update RREQ header
        rt->rt_add(iph->saddr(), godv->rq_id);
        godv->nodePath[godv->length] = my_id_;
        godv->length++;
        cmh->direction() = hdr_cmn::DOWN;
        cmh->addr_type() = NS_AF_INET;
        cmh->last_hop_   = my_id_;
        cmh->next_hop_   = IP_BROADCAST;
      }
      else if(godv->repValid){ //RREP
        //check for intersection
        MobileNode *prevNode = (MobileNode
*)(Node::get_node_by_address(cmh->last_hop_));
        MobileNode *thisNode = (MobileNode
*)(Node::get_node_by_address(my_id_));
        if(prevNode->X() != thisNode->X() &&
prevNode->Y() != thisNode->Y()){
          if(godv->overlayNode[godv->pathLen-1] ==
cmh->last_hop_){ //check Overlay Node list;
```

```
            godv->path[godv->pathLen][0] =
thisNode->X();
            godv->path[godv->pathLen][1] =
thisNode->Y();
            godv->overlayNode[godv->pathLen] =
my_id_;
            godv->pathLen++;
          }
          else{
            godv->path[godv->pathLen][0] =
prevNode->X();
            godv->path[godv->pathLen][1] =
prevNode->Y();
            godv->overlayNode[godv->pathLen] = cmh-
>last_hop_;
            godv->pathLen++;
            godv->path[godv->pathLen][0] =
thisNode->X();
            godv->path[godv->pathLen][1] =
thisNode->Y();
            godv->overlayNode[godv->pathLen] =
my_id_;
            godv->pathLen++;
          }
        }

        cmh->num_forwards()++;
        cmh->addr_type() = NS_AF_INET;
        cmh->last_hop_ = my_id_ ;
        cmh->next_hop_ = godv->nodePath[godv-
>index];
        godv->index++;
        cmh->direction() = hdr_cmn::DOWN;
      }
    }
  }

  if(cmh->ptype() == PT_GPSR){
    struct hdr_gpsr *gh = HDR_GPSR(p);
    switch(gh->type_){
    case GPSRTYPE_HELLO:
      recvHello(p);
      break;
    case GPSRTYPE_QUERY:
```

```
      recvQuery(p);
      break;
    case GPSRTYPE_ROUTEDISC:
      send(p, 0);
      break;
    default:
      printf("Error with gf packet type.\n");
      exit(1);
    }
  }
  else {
    iph->ttl_--;
    if(iph->ttl_ == 0){
      drop(p, DROP_RTR_TTL);
      return;
    }
    forwardData(p);
  }
  done:
   p = 0;
   return;
}
```

A.7. Kode Implementasi *Forward Data*

```
void
GPSRAgent::forwardData(Packet *p){
  struct hdr_cmn *cmh = HDR_CMN(p);
  struct hdr_ip *iph = HDR_IP(p);

  if(cmh->direction() == hdr_cmn::UP &&
     ((nsaddr_t)iph->daddr() == IP_BROADCAST ||
      iph->daddr() == my_id_)){
    sinkRecv(p);
    printf("receive\n");
    port_dmux_->recv(p, 0);
    return;
  }
  else {
    struct hdr_gpsr_data *gdh = HDR_GPSR_DATA(p);
    struct hdr_godv_data *godv = HDR_GODV_DATA(p);

    //check this node
    if(iph->saddr() != my_id_){
      if(godv->index < godv->length-1){
        godv->index++;
      }
      gdh->sx_ = my_x_;
      gdh->sx_ = my_y_;
      gdh->dx_ = godv->path[godv->index][0];
      gdh->dy_ = godv->path[godv->index][1];
    }

    double dx = gdh->dx_;
    double dy = gdh->dy_;

    nsaddr_t nexthop;
    if(gdh->mode_ == GPSR_MODE_GF){
      nexthop = nblist_->gf_nexthop(dx, dy);
      MobileNode *nextNode = (MobileNode
*)(Node::get_node_by_address(nexthop));
      if(nexthop == -1){
              nexthop = nblist_-
>peri_nexthop(planar_type_, -1,gdh->sx_, gdh-
>sy_,gdh->dx_, gdh->dy_);
              gdh->sx_ = my_x_;
```

```
                gdh->sy_ = my_y_;
                gdh->mode_ = GPSR_MODE_PERI;
        }
      }
      else {
        double sddis = nblist_->getdis(gdh->sx_, gdh-
>sy_, gdh->dx_, gdh->dy_);
        double mydis = nblist_->getdis(my_x_, my_y_,
gdh->dx_, gdh->dy_);
        if(mydis < sddis){
                //switch back to greedy forwarding
mode
                nexthop = nblist_->gf_nexthop(dx, dy);
                gdh->mode_ = GPSR_MODE_GF;

         if(nexthop == -1){
           nexthop = nblist_-
>peri_nexthop(planar_type_, -1, gdh->sx_, gdh->sy_,
gdh->dx_, gdh->dy_);
           gdh->sx_ = my_x_;
           gdh->sy_ = my_y_;
           gdh->mode_ = GPSR_MODE_PERI;
         }
        }
        else{ //still perimeter routing mode
                nexthop = nblist_-
>peri_nexthop(planar_type_, cmh->last_hop_, gdh-
>sx_, gdh->sy_, gdh->dx_, gdh->dy_);
        }
      }
      cmh->direction() = hdr_cmn::DOWN;
      cmh->addr_type() = NS_AF_INET;
      cmh->last_hop_ = my_id_;
      cmh->next_hop_ = nexthop;
      send(p, 0);
    }
}
```

A.8. Kode AWK *Packet Delivery Ratio* (PDR)

```
BEGIN {
       sendLine = 0;
       recvLine = 0;
       forwardLine = 0;
}

$0~/^s.*AGT/{
       sendLine++
}

$0~/^r.*AGT/{
       recvLine++
}

$0~/^f.*RTR/{
       forwardLine++
}

END{
       printf"Ratio:%.4f\n",(recvLine/sendLine)*100;
}
```

A.9. Kode AWK *End to End Delay*

```
BEGIN {
       seqno = -1;
       # droppedPackets = 0;
       # receivedPackets = 0;
       count = 0;
}
{
       if($4 == "AGT" && $1 == "s" && seqno < $6) {
              seqno = $6;
       }
       if($4 == "AGT" && $1 == "s") {
              start_time[$6] = $2;
       } else if(($7 == "cbr") && ($1 == "r")) {
              end_time[$6] = $2;
       } else if($1 == "D" && $7 == "cbr") {
              end_time[$6] = -1;
       }
}
END {
       for(i=0; i<=seqno; i++) {
              if(end_time[i] > 0) {
                     delay[i] = end_time[i] -
start_time[i];
                     count++;
              }
              else{
                     delay[i] = -1;
              }
       }
}
for(i=0; i<=seqno; i++) {
       if(delay[i] > 0) {
              n_to_n_delay = n_to_n_delay +
delay[i];
       }
}
       n_to_n_delay = n_to_n_delay/count;
       print n_to_n_delay * 1000 ;
}
```

A.10. Kode AWK Routing Overhead

```
BEGIN{
       recvs = 0;
       routing_packets = 0;
}

{
       if(($1 == "r") && ($7 == "cbr"))
              recvs++;
       if(($1 == "s" || $1 == "r") && ($4 == "RTR")
&& ($7 == "gpsr"))
              routing_packets++;
}

END{
       printf("\nrouting        packets:         %d",
routing_packets);
       printf("\ndata = %d", recvs);
       printf("\noverhead         :         %.3f\n",
routing_packets/recvs);
}
```

A.11. Instalasi NS-2.35

1. Instalasi dependensi yang dibutuhkan
   Sebelum melakukan instalasi NS-2 perlu dilakukan instalasi dependesi yang dibutuhkan NS-2. Perintah untuk melakukan instalasi dependensi adalah sebagai berikut:

   ```
   # sudo-apt-get install build-essenstial autoconf automake
   ```

2. Unduh *file* NS-2
   Unduh *file* NS-2 pada *link* berikut:

   ```
   https://drive.google.com/file/d/0B7S255p3kFXNSGJCZ2YzUGJDVk0/view
   ```

3. Ekstraksi *file* unduhan
   Ekstrak *file* hasil unduhan dengan perintah sebagai berikut:

   ```
   # tar -xvf ns-allinone-2.35_gcc482.tar.gz
   ```

4. Tambahkan *link path*
   Supaya NS-2 dapat dijalankan dimana saja perlu dilakukan pendefinisian *link path* pada .bashrc. Buka *file* .bashrc dengan perintah sebagai berikut:

   ```
   # sudo gedit .bashrc
   ```

   Tambahkan *link path* pada baris paling bawah pada *file* tersebut. *Link path* yang ditambahkan pada .bashrc adalah sebagai berikut:

   ```
   # LD_LIBRARY_PATH
   OTCL_LIB=/home/ubuntu/ns-allinone-2.35/otcl-1.14
   NS2_LIB=/home/ubuntu/ns-allinone-2.35/lib
   X11_LIB=/usr/X11R6/lib
   USR_LOCAL_LIB=/usr/local/lib
   ```

```
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_L
IB:$NS2_LIB:$X11_LIB:$USR_LOCAL_LIB
# TCL_LIBRARY
TCL_LIB=/home/ubuntu/ns-allinone-2.35/tcl8.5.10/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB
# PATH
XGRAPH=/home/ubuntu/ns-allinone-
2.35/bin:/home/ubuntu/ns-allinone-
2.35/tcl8.5.10/unix:/home/ubuntu/ns-allinone-
2.35/tk8.5.10/unix
#the above two lines beginning from xgraph and ending with
unix should come on the same line
NS=/home/ubuntu/ns-allinone-2.35/ns-2.35/
NAM=/home/ubuntu/ns-allinone-2.35/nam-1.15/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

Pada pendefinisian di atas, NS-2 terletak pada direktori /home/ubuntu/ apabila NS-2 terletak pada direktori yang berbeda, ganti /home/ubuntu/ dengan direktori yang benar. Proses instalasi NS-2 akan dilanjutkan setelah dilakukan *patching* protokol GPSR.

A.12. *Patching* Protokol GPSR

1. Unduh *file patch* protokol GPSR
   Unduh *patch* GPSR pada *link* berikut:

   https://drive.google.com/file/d/0B7S255p3kFXNV2ctNFctd3JsZGs/view

2. *Patching* protokol GPSR pada NS-2
   Letakkan *file patch* pada direktori NS-2. Selanjutnya lakukan *patching* dengan perintah sebagai berikut:

   ```
   # patch -p0 < gpsr-Keliu_ns235.patch
   ```

3. Modifikasi Makefile
   Modifikasi *file* Makefile karena adanya modifkasi protokol gpsr dengan adanya penambahan *file* godv_rtable. Lakukan pendefinisian godv_rtable.o pada *file* Makefile di bawah gpsr/gpsr.o.

   ```
   388  gpsr/gpsr.o
   389  gpsr/godv_rtable.o
   ```

4. *Install* NS-2
   Lakukan instalasi NS-2 dengan perintah sebagai berikut:

   ```
   # ./install
   ```

BIODATA PENULIS

Kevin Arditya dilahirkan di Kota Surakarta pada tanggal 31 Desember 1994. Penulis adalah anak pertama dari dua bersaudara. Pendidikan yang pernah ditempuh penulis di mulai dari SD Al Firdaus (2001-2007), SMP Negeri 4 Surakarta (2007-2010), SMA Negeri 1 Surakarta (2010-2013). Pada tahun 2013, penulis diterima di Strata Satu Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya melalui jalur undangan SNMPTN. Di Jurusan Teknik Informatika, penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK). Penulis dapat dihubungi melalui *email* kevin.arditya@gmail.com